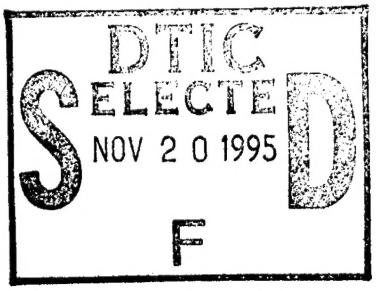
WL-TR-95-7034

Optical Spatial Filter Development

Robert R. Kallman

Department of Mathematics
University of North Texas
Denton TX 76203-5116



Contract No. F08635-90-K-0102

JUNE 1995

FINAL REPORT FOR PERIOD SEPTEMBER 1990 - DECEMBER 1994

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION IS UNLIMITED

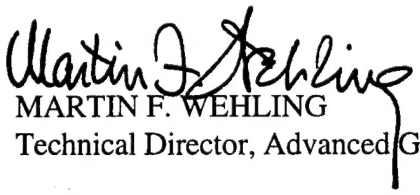
WRIGHT LABORATORY, ARMAMENT DIRECTORATE
Air Force Materiel Command ■ United States Air Force ■ Eglin Air Force Base

NOTICE

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely Government-related procurement, the United States Government incurs no responsibility or any obligation whatsoever. The fact that the Government may have formulated or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication, or otherwise as in any manner construed, as licensing the holder, or any other person or corporation; or as conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

This technical report has been reviewed and is approved for publication.

FOR THE COMMANDER



MARTIN F. WEHLING
Technical Director, Advanced Guidance Division

Even though this report may contain special release rights held by the controlling office, please do not request copies from the Wright Laboratory, Armament Directorate. If you qualify as a recipient, release approval will be obtained from the originating activity by DTIC. Address your request for additional copies to:

Defense Technical Information Center
8725 John J. Kingman Road, Suite 0944
Ft. Belvoir VA 22060-6218

If your address has changed, if you wish to be removed from our mailing list, or if your organization no longer employs the addressee, please notify WL/MNGA, Eglin AFB FL 32542-6810, to help us maintain a current mailing list.

Do not return copies of this report unless contractual obligations or notice on a specific document requires that it be returned.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	June 1995	Final Technical Report Sep 90 - Dec 94

4. TITLE AND SUBTITLE
Optical Spatial Filter Development

5. FUNDING NUMBERS
C: F08635-90-K-0102
PE: 62602F
PR: 2068
TA: 30
WU: 21

6. AUTHOR(S)
Robert R. Kallman

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
University of North Texas
Mathematics Department
PO Box 5116
Denton TX 76203-5116

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Advanced Guidance Division
Wright Laboratory Armament Directorate
101 W. Eglin Blvd.
Eglin AFB FL 32542-6810

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
WL-TR-95-7034

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited. ~~Requests for this document shall be referred to WL/MNGA, Eglin AFB Florida 32542-6810~~

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The objective of this program was the design of optical spatial filters. Among the topics discussed in this report are: the proper role of correlation in target recognition and discrimination; noise considerations for the objective function used in designing spatial filters; correct computer modeling of the correlation process; a completely new approach to the design of general spatial filters as a constrained optimization question in all free parameters (all amplitudes and phases); the design of spatial filters whose entries are either zero or a continuous phase as an optimization question; the optimal discretization of spatial filters whose entries are either zero or a continous phase; the need and sufficiency for spatial filters whose entries are either zero or an nth root of unity; the problems of target/background contrast and target translation invariance with respect to background; intensity phase-encoding of imagery and its advantages; zero-mean intensity phase-encoding of imagery and its advantages in solving the background problems; and a new edge enhancement/binarization procedure based on phase-encoding and local Wigner transforms.

A secondary purpose of this document is to be a user guide to the software created during this effort for the design of spatial filters. A goal of this effort has been a drastic refinement, extension, and extrapolation of spatial filter design codes in the parallel programming language occam 2 for use in inexpensive transputer based parallel processing systems. Spatial filter design on an occam 2/transputer based system is now approximately 250 times faster than on a FORTRAN/Vax system.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Optical correlation, optical pattern recognition, optical spatial filters	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	SAR

PREFACE

This program was conducted by Robert R. Kallman, Department of Mathematics, University of North Texas, P.O. Box 5116, Denton, Texas 76203-5116 under contract F08635-90-K-0102 with Wright Laboratory Armament Directorate, Eglin Air Force Base, Florida 32542-5434. Dr. Dennis H. Goldstein, WL/MNGA, managed the program for Wright Laboratory. The work was conducted during the period from September 20, 1990 through September 30, 1994.

TABLE OF CONTENTS

I INTRODUCTION

The importance and efficacy of correlation and matched filtering well known in one-dimensional radar signal processing (Skolnik, Reference 1). There is therefore an understandable interest in studying correlation and spatial filtering, at least as a potentially useful tool if not a cure-all, for two-dimensional target recognition and discrimination. Addressable optical correlators might well find uses in military and industrial pattern recognition because of their potential high frame rate, compact size, low weight, and low power requirements. However, even if all hardware design considerations can be overcome to create such optical signal processing devices with every desirable characteristic, their performance will be only as good as the spatial filters which are used in the correlation process. Good spatial filters are the key software designed ingredient needed to make such devices useful.

The main purpose of this document is to give a concise but comprehensive discussion of the author's philosophy of and approach to the design of spatial filters. Section II lists the objectives of the current program. A description of the imagery used in this effort is given in Section III. The proper role of correlation in target recognition and discrimination is discussed in Section IV. Section V gives nuances of the computer simulations designed to test the performance of spatial filters. Section VI gives a small amount of empirical evidence to show that simple matched filters are not adequate for many correlation purposes. The objective function used to measure the performance of a spatial filter is stated and discussed in Section VII. What noise considerations, if any, should be incorporated into the objective function is discussed in detail and the usual signal-to-noise ratio (SNR) is criticized. The formulation of spatial filter design as a mathematical optimization question is given in Section VIII. Section IX discusses in detail the algorithm to be used to optimally design spatial filters whose entries are either zero or complex numbers of modulus one. Algorithms used to optimally design the much more complicated general spatial filters are given in Section X. An algorithm to choose the region of support in a filter whose entries are either zero or complex numbers of modulus one is given in Section XI, and an algorithm to optimally discretize a filter whose entries are either zero or complex numbers of modulus one into a filter whose entries are either zero or the nth roots of unity is given in Section XII. Intensity phase-encoding

of byte images is discussed in Section XIII. A basic problem in all spatial filtering, the background contrast problem, is recalled in Section XIV. The lack of translation invariance for targets in nontrivial backgrounds is pointed out in Section XV. A very tentative solution to the problems discussed in the previous two sections is given in Section XVI. A new edge enhancement algorithm based on phase-encoding and local Wigner transforms was developed during the course of this work by Elaine Pettit and the author. This algorithm is described in Section XVII, where it is contrasted with two other popular edge enhancement techniques. Conclusions are listed in Section XVIII and recommendations are listed in Section XIX.

A secondary purpose of this document is to be a user guide to the software created during this effort for the design of spatial filters. Appendix A gives an introduction to Appendices B-G. Appendix H gives a few highlights of the image processing and spatial filter simulation software delivered to Wright Laboratory. The author's computing environment is described in Appendix I. The computational practicality of the author's algorithms is discussed in detail in Appendix J.

II THE OBJECTIVES OF THE PROGRAM

Circa 1984 the author formulated the design of phase-only filters for optical correlators as a complicated nonlinear nondifferentiable optimization question in a very large number of free parameters. In a previous effort he had also found fairly effective mathematical algorithms for the solution of this class of optimization problems and had implemented them in FORTRAN for use in a VAX computing environment. In this previous effort he had also devised algorithms and implemented them in FORTRAN to optimally binarize phase-only filters. An important goal of this effort has been a drastic refinement, extension, and extrapolation of these algorithms and their implementation in the parallel programming language occam 2 for use in inexpensive transputer based parallel processing systems. This has been accomplished. Spatial filter design on an occam 2/transputer based system is now approximately 250 times faster than on a FORTRAN/VAX system. There has been a similar increase in speed for optimally discretizing a phase-only filter into not just two but an arbitrary number of equally spaced phase states. In extensive recent tests total design times for hundreds of optimized binary filters with large training sets invariably ranged between ten and fifteen minutes for each filter. The algorithms devised by the author for phase-only filter design are described in Section IX. The algorithms for optimal discretization are described in Section XII.

The author proposed a method for designing a general spatial filter as a constrained optimization question. This proposed method makes full use of all the free parameters in the design problem. An important goal of this effort was to find efficient algorithms to solve this design optimization question and to implement them in computer codes in the parallel programming language occam 2 for use in inexpensive transputer based parallel processing systems. This has been accomplished. The algorithms devised by the author for phase-only filter design are described in Section X.

Another goal of this effort was to create an algorithm to determine in a reasonable manner the region of support in a spatial filter whose entries are either zero or a complex number of modulus one. This has been accomplished. The algorithm to solve this problem is sketched in Section XI. This algorithm makes essential use of the general spatial filter described in

the previous paragraph. Indeed, it was with an eye toward this application that partially motivated the author's efforts in general spatial filter design.

An assigned task of this effort was to deliver to Wright Laboratory a user friendly version of the author's spatial filter design codes together with a comprehensive User Guide. These items have been delivered to Wright Laboratory. The User Guide to the author's spatial filter design codes are given in Appendix A — Appendix G.

Processing of target imagery and testing the performance of spatial filters necessitated the creation of a variety of graphics programs. These were written completely from scratch to drive a SONY GDM-1953 graphics color monitor. These graphics programs have proven invaluable in viewing, manipulating, edge enhancing, and binarizing imagery and in simulating the optical correlation process in great detail. An occam 2 program to drive a transputer based framegrabber system was also created. The characteristics and capabilities of these graphics software tools are sketched in Appendix H.

Circa 1985 the author noted two difficulties in spatial filtering which apparently had not been noted up to that time. The first difficulty arises from variations in the target/background contrast. The second arises when a target in a nontrivial background is translated with respect to the background. These difficulties are discussed in Section XIV and Section XV. A goal of this effort was to devise a reasonably effective solution to these two problems while preserving the advantages of correlation. This has been accomplished. A tentative solution is discussed in Section XVI.

Furthermore, a large part of the labor during this effort consisted of the design and construction of large numbers of spatial filters which were forwarded to Wright Laboratory. Thousands of spatial filters were delivered during the course of this effort. The author believes that all requests for spatial filters were answered rapidly and that the resulting spatial filters more than met all design requirements.

III THE TRAINING SET IMAGERY

The Wright Laboratory Armament Directorate supplied virtually all of the imagery used in this study. Some of the imagery used was from an older vintage and was used in a previous effort. Most of this older imagery was supplied on several VAX-compatible computer tapes. The imagery was produced by a visible light intensity digitizing camera from models on a target cloth background from a small variety of depression angles (20, 25, 30, 35, and 40 degrees) and over a wide swath of azimuth angles at one degree increments (-45 — 180 degrees). Targets in this older imagery consisted of M1, M113, M48, M60, T62, and T72 images. Each original image was originally 128×128 pixels in size with the intensity of each pixel scaled to be an integer between 0 and 255 and stored in byte format. Each of these older targets needed to be separated from its background. To facilitate this the targets were uniformly brightly illuminated so that the background intensity off the target cloth was digitized to be 255. The target then was simply extracted from the background by setting all pixels with intensity value 255 equal to 0. What remained was a target in a background consisting of pixels with the intensity value 0. This byte imagery was then further processed via some simple occam 2 programs created by the author in order to eliminate a few stray very low intensity and very high intensity pixels and to center the remaining target about its nonzero centroid. Figure 1 shows a typical Old Series (OS) M48 (more specifically m48d20.090 (OS)) in a blank background. Here (and later) this figure is a black and white PMT of a color photograph of the target displayed in false colors to make its variations more prominent. The notation used here and later for the Old Series targets is self explanatory — "m48" connotes that the target was an M48; "d20" connotes that the depression angle was 20 degrees, "090" connotes that the azimuthal angle was 90 degrees, and "(OS)" connotes that this is an Old Series image.

The New Series imagery consisted of real target images originally located on a turntable at Range C-52 at Eglin AFB. Captain Eric P. Augustus (EPA) of Wright Laboratory Armament Directorate was in overall charge of this work. The target list included M48 and M60 tanks and C35 and M35 trucks. The notation used here is a bit nonstandard — C35 truck images really are M35 truck images with a canopy, whereas the M35 images were uncovered. Images

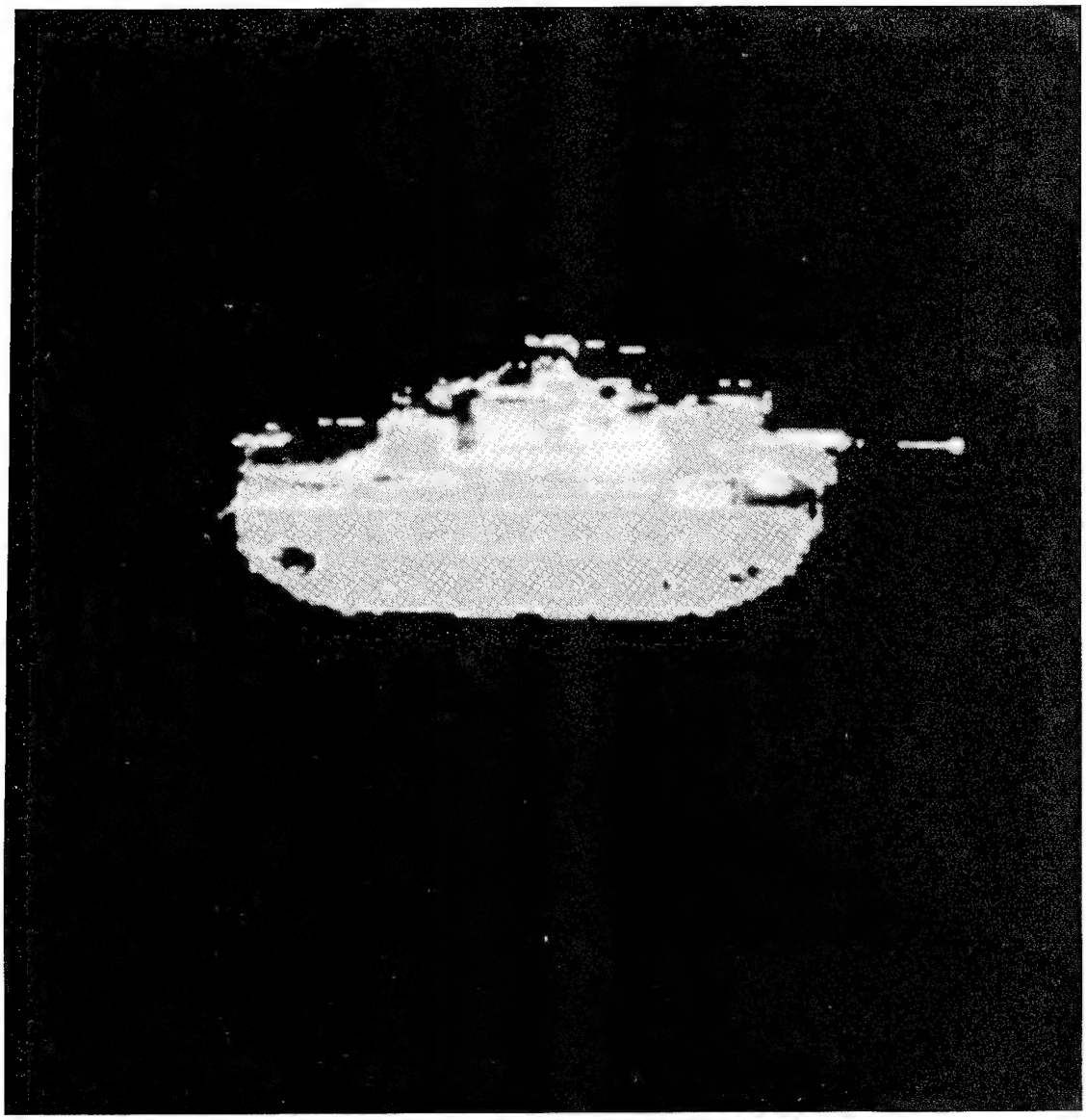


Figure 1. m48d20.090 (OS)

of each target were acquired at depression angles of 20 — 50 degrees in 5 degree increments and at 360 azimuthal degrees in one degree increments. The images were created using a TV camera mounted on a tower located at the range. The TV camera output was digitized using a Transtech TTG-F framegrabber transputer module on a transputer module motherboard located in a PC. The author initially wrote an occam 2 program to drive the framegrabber. This occam 2 framegrabber program of the author's was very extensively upgraded and extended by EPA, and it was this latter program which was actually used to digitize the TV signal. Each range image was originally 256×256 pixels in size with the intensity of each pixel scaled to be an integer between 0 and 255 and stored in byte format. The actual target occupied a relatively small portion of each of these 256×256 images. A 128×128 byte subimage containing the actual target was extracted from each of the 256×256 range images using an extension of the occam 2 gp1 graphics codes (described in Appendix H). This extension again was created by EPA. Figure 2 shows the 128×128 turntable image of m48d25.090 (NS). Here "(NS)" connotes that this is a New Series image. These 128×128 turntable byte images were processed in turn by workers at Wright Laboratory using the PC-based Matisse Windows program by Fauve to extract the targets from their background. Clean target imagery unencumbered by a background is needed for training sets in spatial filter design. Figure 3 shows the 128×128 clean target image m48d25.090 (NS).

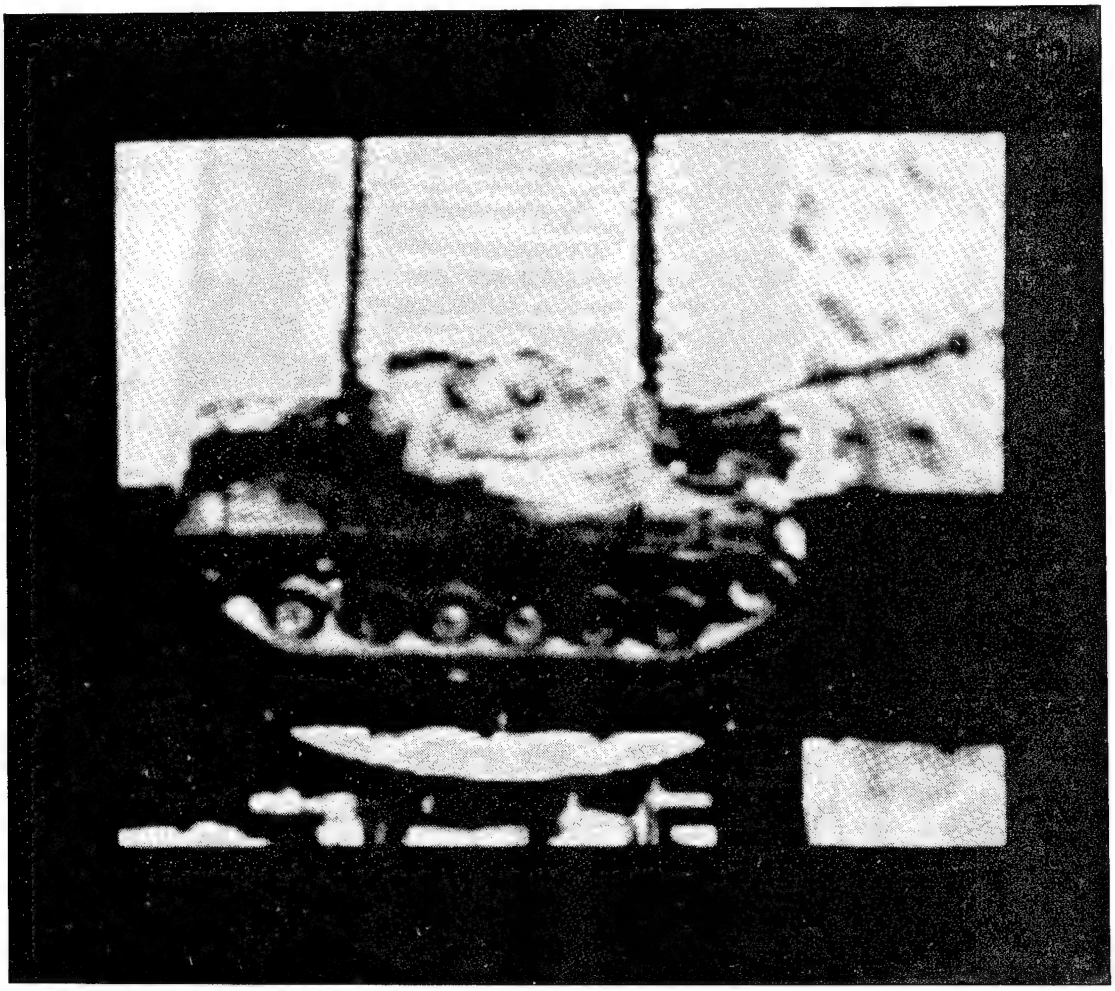


Figure 2. m48d25.090 (NS) on a Turntable at Range C-52

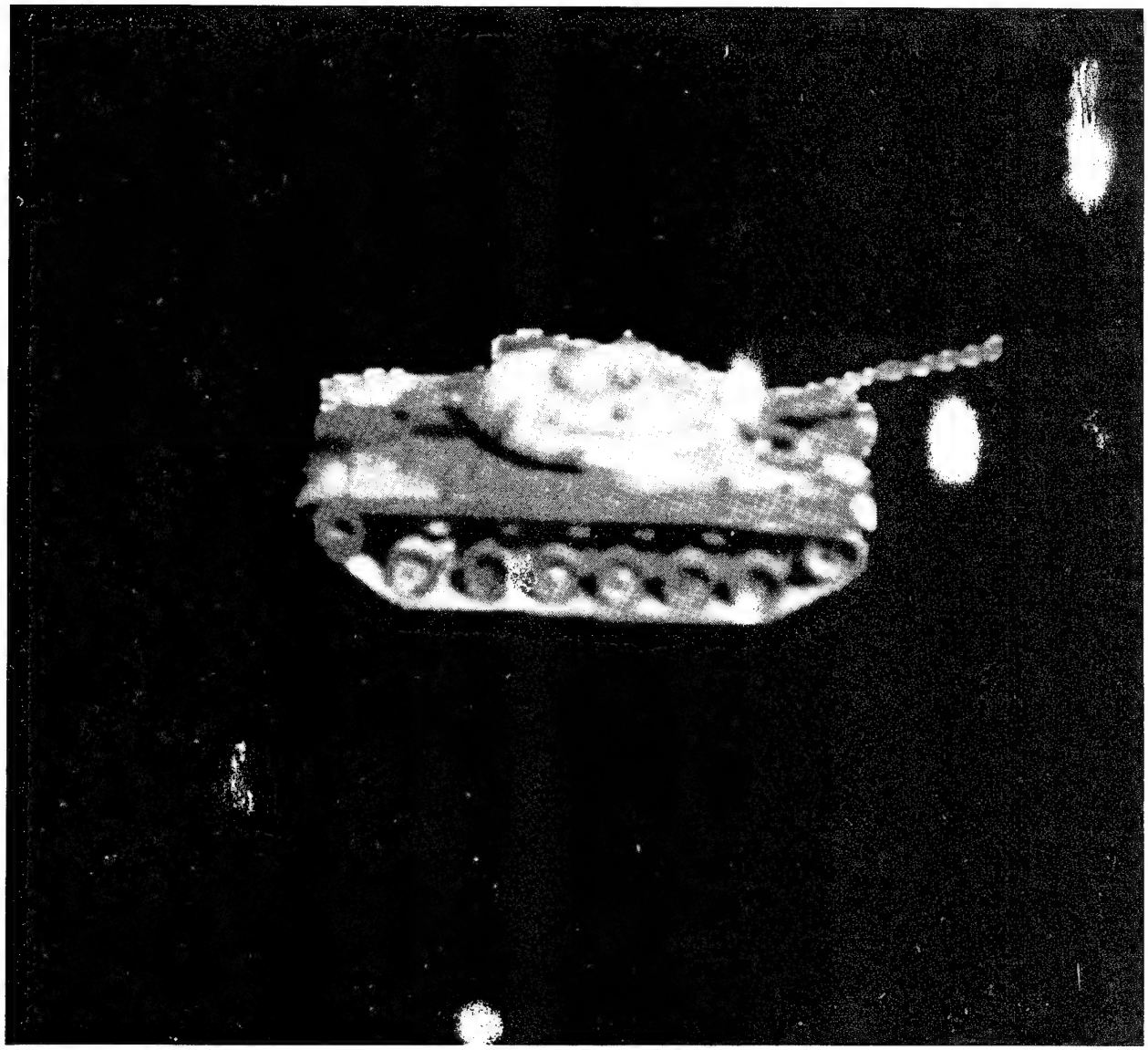


Figure 3. m48d25.090 (NS)

IV THE ROLE OF CORRELATION IN TARGET RECOGNITION AND DISCRIMINATION

One-dimensional radar signal processing is a well established discipline in which the importance and efficacy of correlation and matched filtering are well known. This strongly suggests that correlation and spatial filtering should be studied for two-dimensional target recognition and discrimination, at least as a potentially useful tool if not a cure-all. This is also strongly suggested by the following general considerations.

First, if one is fortunate or intelligent enough to have in hand a few simple unexceptional features enabling one to tell one (potential) target or class of (potential) targets from another or from images not containing targets, then one should use them. Examples of such unexceptional features might be the width and/or height of an object. Note that even here correlation might prove useful if the black box performing the correlation has a sufficiently high frame rate. In any case such simple considerations might prove useful in picking target-like objects out of a large and complicated image.

Next, in lieu of or subsequent to the first step, one has little choice but to use detailed a priori knowledge about the true and/or false target images themselves, the only knowledge which is known to be absolutely correct. The best use to which such absolutely correct target information can be put is to incorporate it into a compact library of carefully designed spatial filters for use in a digital or optical correlator. This should eliminate the perhaps dubious practice of using a list of human selected features for target identification and discrimination. Implicit in this discussion is a belief that target recognition based on a priori statistical assumptions about noise and/or false targets is methodologically and philosophically unsound and probably a misguided waste of time. A simple gedanken experiment, which should be kept in mind when reading the rest of this report, will suffice to illustrate these ideas.

Suppose that one has an enormous table densely covered with a wide variety of nonoverlapping keys at random orientations. Suppose further that one has a particular key in mind and one would like to find every key on the table, if any, which matches this particular key. Corresponding to the first step discussed above, one could quickly and easily eliminate from consideration any key on the table which was not approximately the same color, length,

width, and height of the desired key. There still might be a great many keys left on the table after this initial screening. What should be done next? One were adopt the philosophy of statistical target recognition, one would probably assume that the serrations on the false keys satisfied some fixed probability distribution (with an explicit, closed, analytical form, no less!) with, say, seventeen free parameters which are to be determined by taking many, many samples of false keys. Then any key whose serrations satisfy the now explicitly determined distribution to a reasonably good fit would be removed from the table. There still might be many keys left on the table, most of which might be false. Ideas similar to statistical target recognition were popular in biology in the 1930's under the rubric of the logistic law of growth. Feller (Reference 2) effectively demolished the scientific validity of such ideas and they lost popularity. A similar fate should probably befall statistical target recognition.

A better idea would be to have in hand a model of the key one desires and to quickly and successively compare this model to each key on the table. This is nothing but correlation.

V SIMULATION CONSIDERATIONS

The simulations to be described in the following sections were done in a manner to model reality as closely as possible. In computer simulations the entire plane first of all is replaced by a square centered at (0,0). The larger the square the better the approximation to the plane in the general neighborhood of (0,0). This square is made into a periodic pattern in the plane by identifying opposite edges. Such a periodic pattern in the plane can be viewed as one such pattern completely occupying the surface of a doughnut. Such a doughnut locally replicates the plane well at points not too remote from (0,0). This continuous doughnut is then replaced by a discrete doughnut periodic along each axis. If the discretization is fine enough, then the discrete doughnut is itself a reasonable approximation to the plane at points not too remote from (0,0). The points on the doughnut usually are identified with the entries in a suitable two-dimensional matrix. Images on the plane are locally approximated by functions on this discrete doughnut, and usually represented in matrix form, two-dimensional integrals are replaced by doubly-indexed finite sums over two-dimensional matrix entries, and two-dimensional Fourier transforms are replaced by two-dimensional finite Fourier series. The latter are invariably evaluated using the two-dimensional fast Fourier transform algorithm.

One gross feature of a real correlator is that the input device is bounded in size. Therefore when a real correlator computes the Fourier transform of an input, it in reality is computing the Fourier transform of an object in an aperture in an infinite plane all of whose values outside the aperture are zero. Suppose that the input consists of an array of intensities. The correlator computes the Fourier transform of the amplitude (not the intensity -c.f. Goodman, Reference 3) of the input image. Now in reality all targets are in nontrivial backgrounds. Therefore, the correlator computes the Fourier transform of a function which is nonnegative (and very often everywhere strictly positive) inside the aperture and which is zero outside. This is approximately true even for binarizations of realistic imagery. This holds since all correlators using spatial light modulators do not have a true zero state. Hence, as its first step, to a first approximation the correlator computes the Fourier transform of a square aperture. Care is needed in spatial filter design and simulation to guarantee that this important aperture box effect is modeled correctly. Consider the two simplest computer

models for 128×128 input images. In the first computer model the input plane and the Fourier transform pl are 128×128 in size. In this first model 128×128 discrete Fourier transforms (DFTs) are computed with the output placed in the 128×128 Fourier plane. Let f be the 128×128 byte array whose entries are all one. Here and in later sections let $\mathcal{F}$ be the Fourier transform operator on functions and the DFT operator on arrays. Figure 4 is a display of $|\mathcal{F}(f)|$. It is, of course, essentially a delta function at the origin, as a simple explicit computation shows. In the second computer model the input plane and the Fourier transform plane are 256×256 in size. All of the entries in the input plane are set equal to zero except those entries in the central 128× portion, into which is placed the 128×128 image of interest. This at least captures som vor of an aperture in a zero background. Let g be that 256×256 input plane into which the previous constant 128×128 image f is centered. Figure 5 is a display of $|\mathcal{F}(g)|$. The read can decide for himself from a glance at Figure 4 and Figure 5 which of these two computer models captures the essence of the Fraunhofer diffraction pattern of a square aperture. Therefore, in the simulations that are discussed in this report all Fourier transforms will be 256×256 in size unless explicitly stated otherwise. All imagery will be 128×128 in size and will be centered in th iddle of a 256×256 input array, whose other entries are set equal to zero.

In this study all spatial filters are 128×128 in size. It is an open question as to how optimize the scaling of this spatial filter in the Fourier plane of an optical correlator. If the spatial filter is implemented in a fixed size spatial light modulator (SLM), this scaling can be accomplished by changes in the lenses, laser wavelength, and other components of the optical system. Consider the extreme cases. If the scaling is very small, then only the lowest spatial frequencies of the target image will influence the correlation plane. If the scaling is very large, then all of the essential spatial frequencies of the target will be located in only a few cells of the SLM and virtually no filtering will be done. In general, there is a certain constant tension in choosing the scaling between the need to do a thorough amount of filtering on given set of spatial frequen and the need to include the essential spatial frequencies of the target in the filtering p ss. In practice, however, the manufacturers of correlators seem to have settled on the nyquist scaling in the Fourier plane. In this full nyquist scaling, if the Fourier transform planes are 128×128 in size, then the 128×128

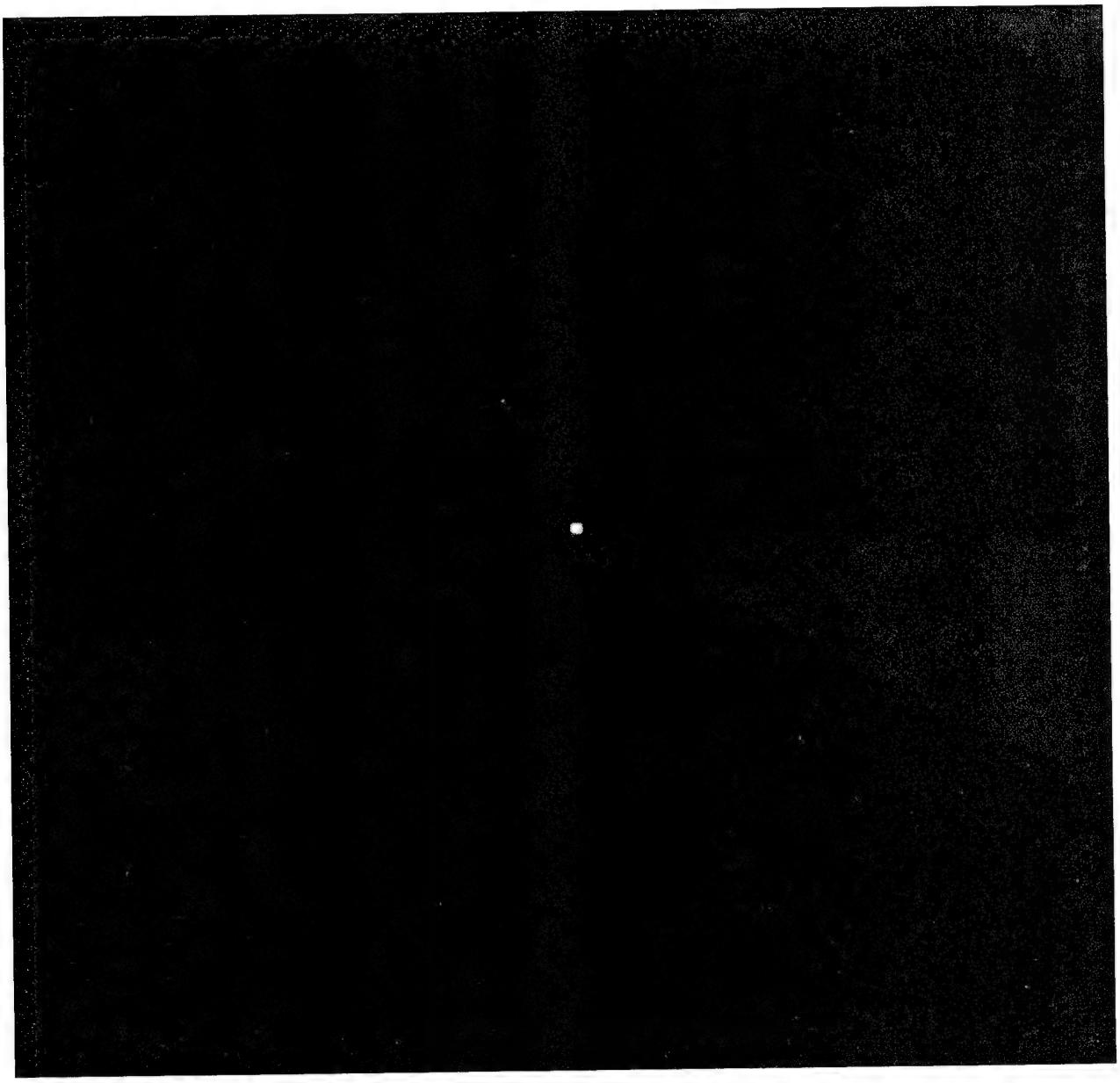


Figure 4. $|\mathcal{F}(\mathrm{f})|$

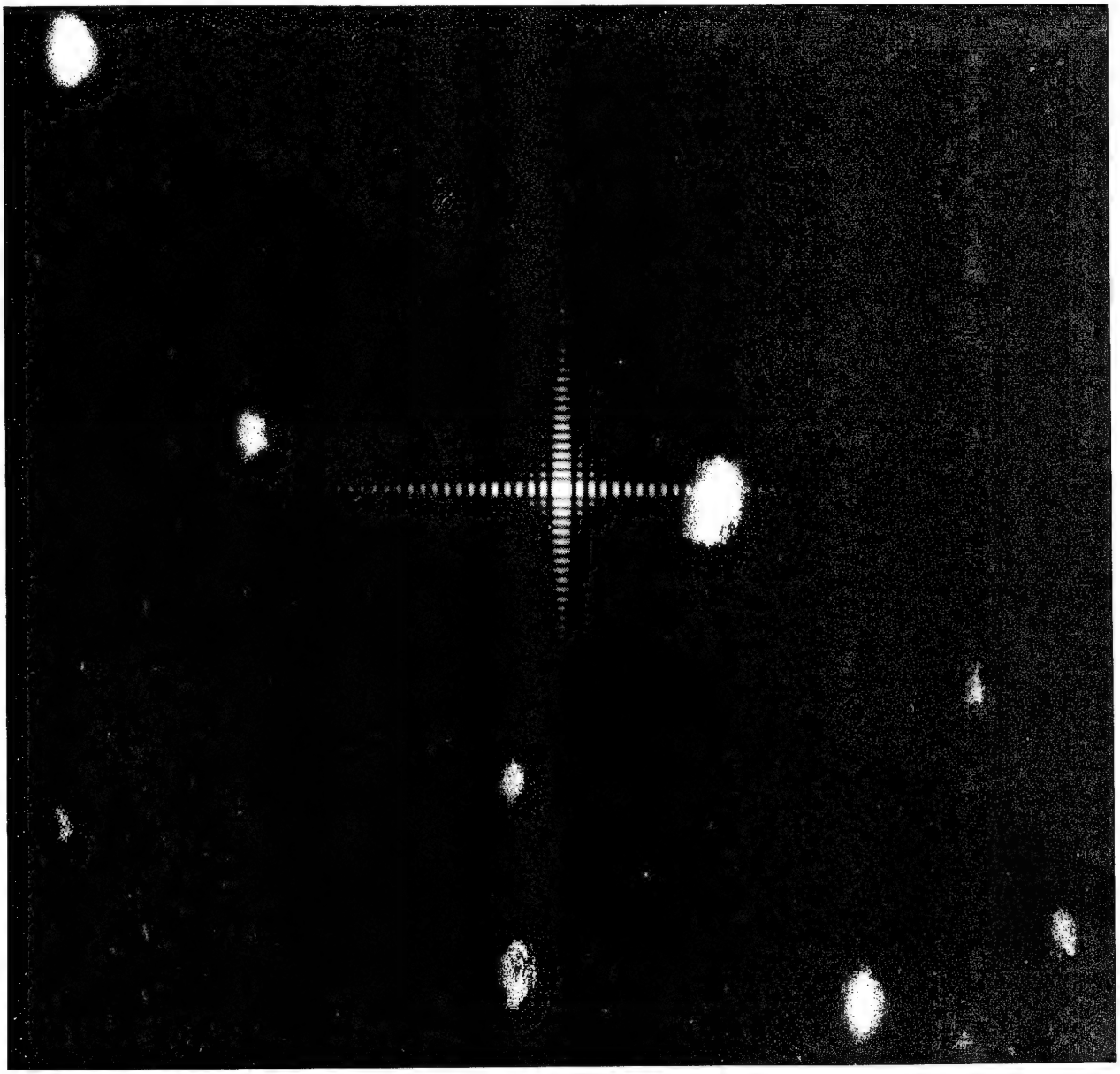


Figure 5. $|\mathcal{F}(g)|$

spatial filter entries are scaled to be in a one-to-one correspondence with the Fourier plane entries. Recall the relation for one dimensional DFTs: $N\Delta x\Delta k = 2\pi$. Here N is the number of samples of a function to be Fourier transformed, Δx is the physical step size between samples, and Δk is the step size in Fourier transform space (this relation might become a bit more familiar if one lets $k=2\pi\omega$. From this one sees that if Δx is constant and $N\rightarrow 2N$, then $\Delta k\rightarrow \Delta k/2$. This tells us that if $128\rightarrow 256$ and the physical cell size is constant (the 128×128 input image is zero padded into the 256×256 input array), then each cell of the 128×128 spatial filter must be scaled up by a factor of 2, i.e., each entry in the spatial filter will cover a 2×2 cell in the Fourier plane. This was done consistently throughout the spatial filter design and simulation work conducted in this effort.

The careful reader may now ask the following question. If zero padding is needed in the input plane to account for the aperture box effect in the input plane, don't we also need to do a similar zero padding in the filter plane before using a Fourier transform to pass to the correlation plane? After all, isn't a typical SLM a square bounded array? Empirically, to a first approximation, the answer is no. The spatial filters which are constructed in this effort are not arrays of positive numbers, but in general are arrays of complex numbers with a relatively small average. This is necessary if they are to discriminate against false targets. The aperture effect in the filter plane is therefore rather small.

Perhaps a final note should be made about some small but important nuances, including biasing considerations, which arises in computer simulations and when computing DFTs. Note that a square with an even number of rows and columns has no natural center. It is assumed that the center (0,0) of the input plane corresponds to the (129,129) entry of the 256×256 matrix. This forces the (65,65) entry of the 128×128 subarray representing the input image to correspond to (0,0) in the input plane. Similar precautions are taken in the Fourier plane. All arrays must be cyclicly biased when computing DFTs so that the origin in the input plane and the correlation plane and the lowest spatial frequency in the filter plane correspond to the (0,0)-th entry in Fourier transform arrays.

VI WHY SIMPLE MATCHED FILTERS ARE INADEQUATE

The role of matched filters is well known in one-dimensional radar signal processing (Skolnik, Reference 1). There is therefore a natural interest in studying the use of correlation and spatial filtering for two-dimensional target recognition and discrimination.

Simple examples show that matched filters have a variety of perhaps surprising defects and are not a panacea in target recognition and discrimination. First of all, their correlation signals are not very sharp. Figure 6 is a top-down view of the correlation plane of the matched filter for m48d20.090 (OS) vs. its training set image, viz., m48d20.090 (OS) itself. Here and in the rest of this report the matched filters are normalized so that their largest amplitude is one. Matched filters can also get confused by nontrivial changes in the contrast between target and background. Figure 7 shows m48d20.090 (OS) in a background of intensity 128 and Figure 8 shows a top down view of the correlation plane of the matched filter used in Figure 6 versus the image in Figure 7. The correlation peaks off the target are almost as high as the correlation peaks on the target. Matched filters are very bulky since an N×N matched filter in general will require $8N^2$ bytes to store. Finally, a useful library of matched filters would need to be huge, perhaps 10^5-10^6 in size, as suggested in a National Academy of Sciences study. Given that a practical correlator would need to process 30 — 100 input images per second, one instantly sees that one would need a very high frame rate correlator even if one has an algorithm to recursively use only a small fraction of the filter library on each input image. Therefore, without even examining their ability to discriminate against false targets, matched filters have a large number of negative attributes which would seem to preclude their practical use in realistic target recognition and discrimination.

In recent years many of these problems associated with matched filters have tentative solutions. The first step was the introduction of phase-only matched filters by Horner and Gianino (Reference 4). Generalizations of this concept will be discussed in detail in the next section. Horner and Gianino were apparently motivated to invent the phase-only matched filter by light budget considerations in optical correlation. The phase-only filter appears to have many other benefits besides making efficient use of light in an optical correlator. For example, Figure 9 shows a top down view of the correlation plane for the phase-only matched

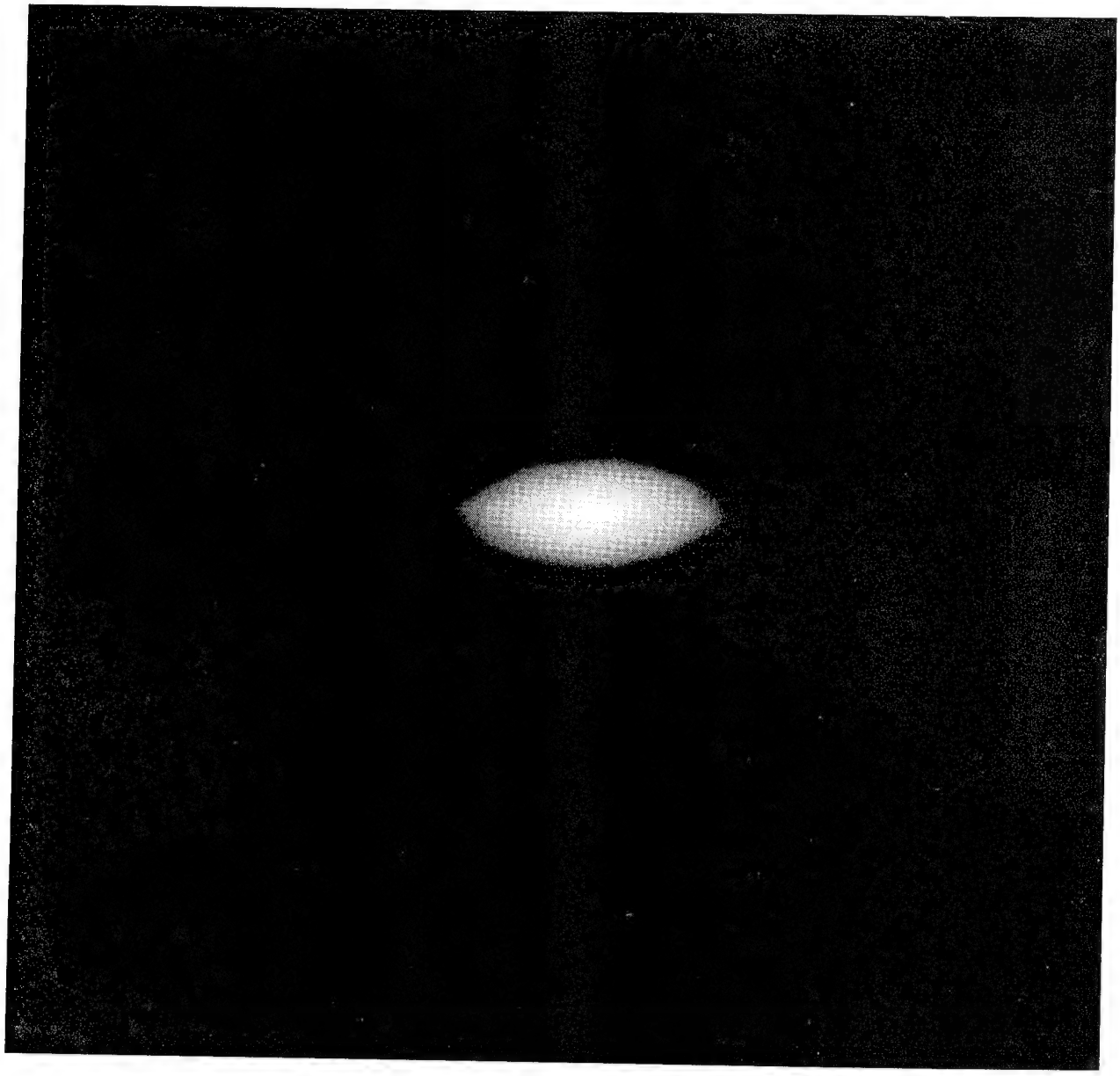


Figure 6. Correlation Plane Top Down View of Matched Filter for m48d20.090 (OS) vs. m48d20.090 (OS)

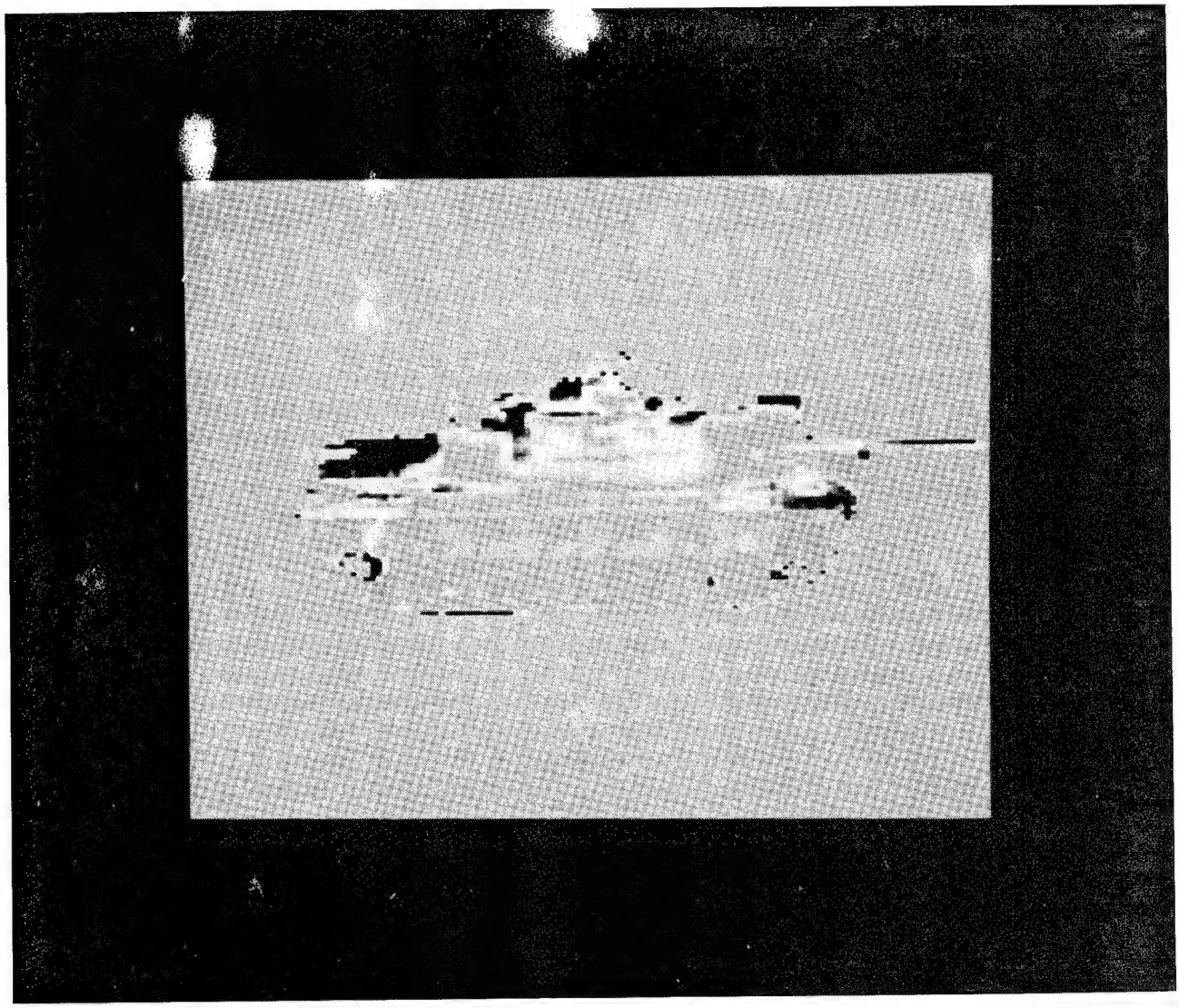


Figure 7. m48d20.090 (OS) in a Background of Intensity 128

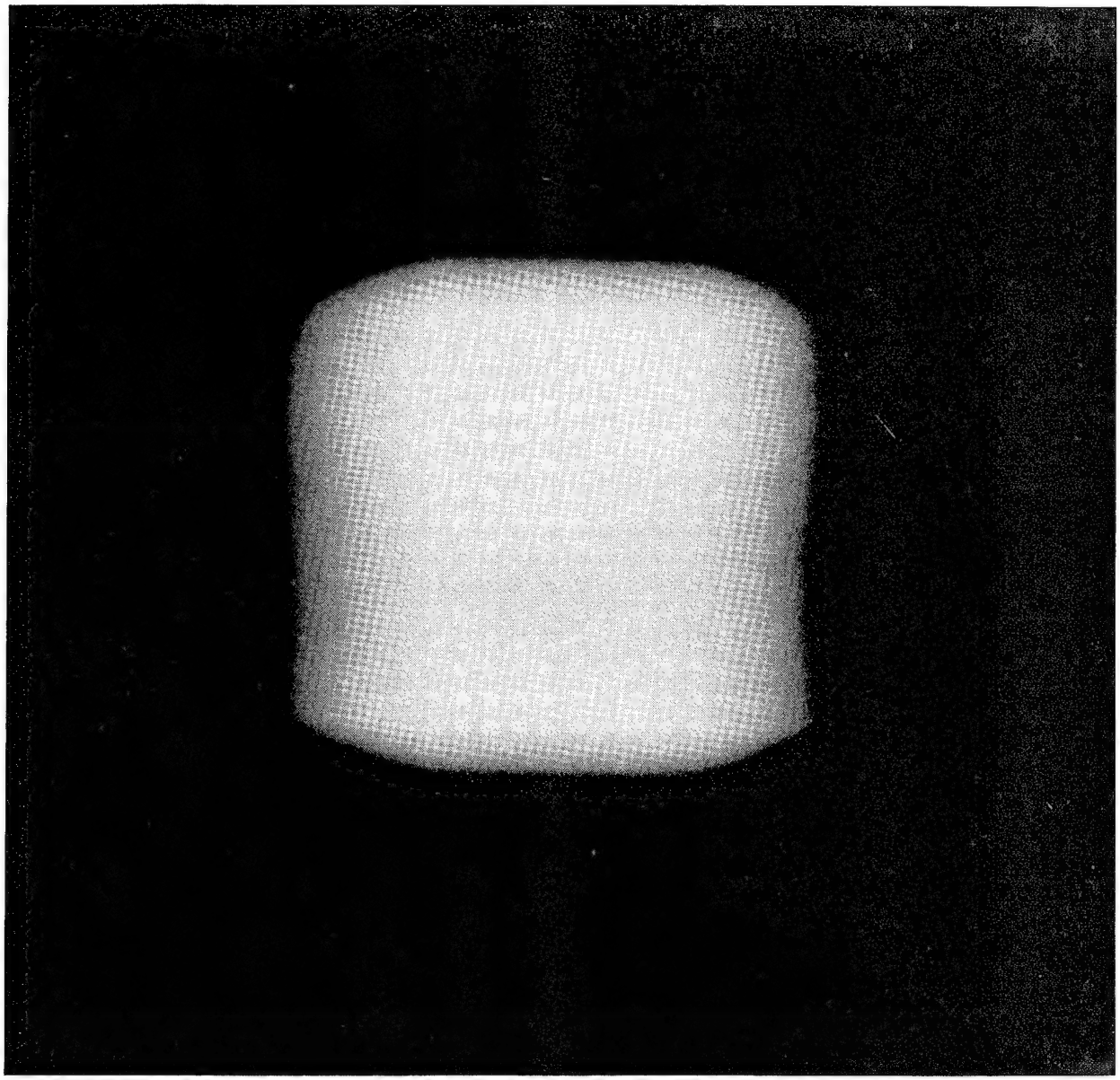


Figure 8. Correlation Plane Top Down View of Matched Filter for m48d20.090 (OS) vs. m48d20.090 (OS) in 128 Intensity Background

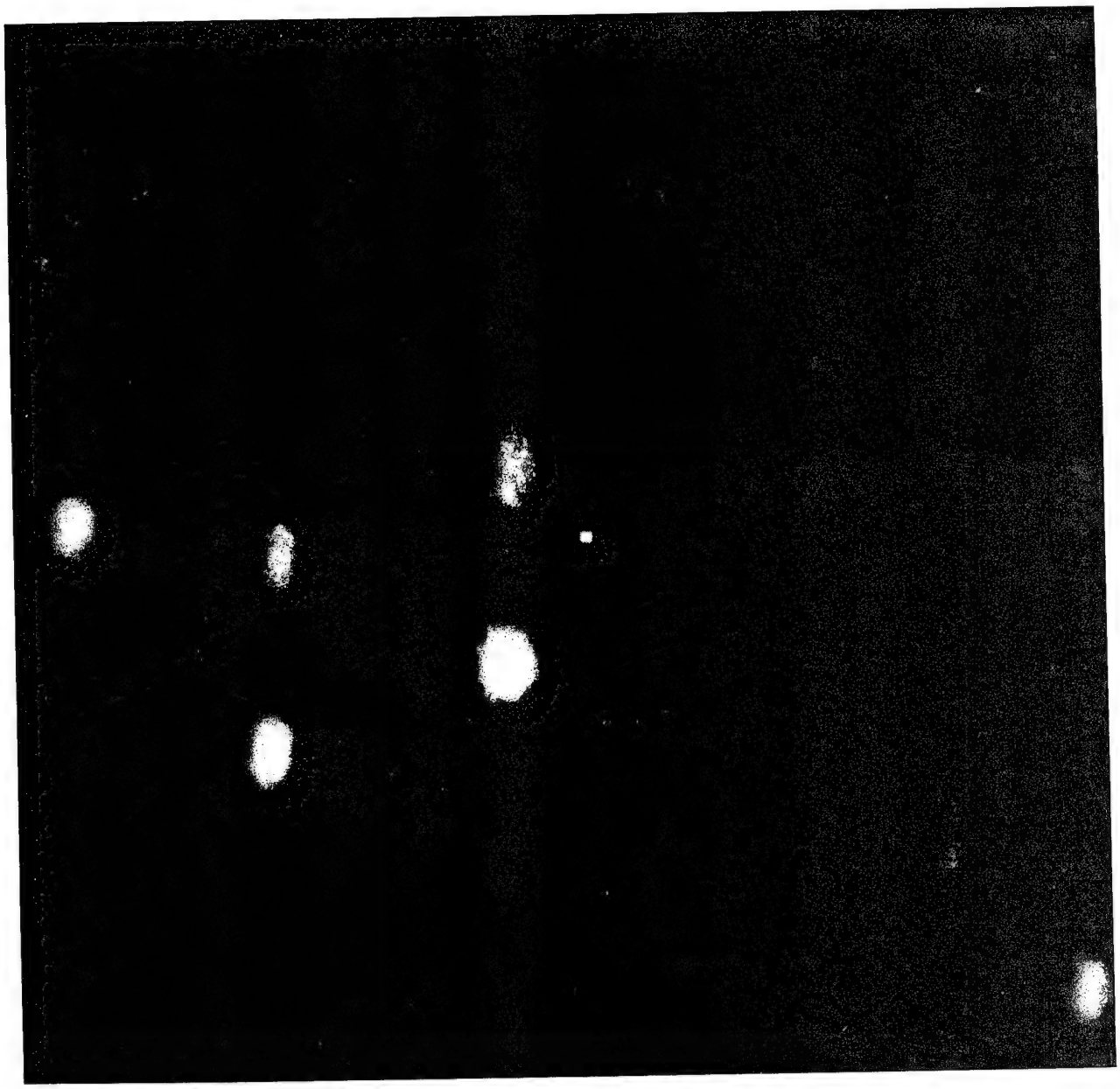


Figure 9. Correlation Plane Top Down View of Phase-Only Matched Filter for m48d20.090 (OS) vs. m48d20.090 (OS)

filter with m48d20.090 (OS) as its training set image vs. m48d209.090 (OS). There is a very sharp recognition spike. Another benefit of the phase-only filter concept is that appropriate generalizations of them can be closely approximated by spatial filters with a small number of discrete states. These discrete state filters can be stored in a much more compact manner (in $N^2/2$ bytes or fewer for N×N filters whose entries are either zero or a 15th root of unity, for example) than matched filters. Algorithms to perform good discretizations are discussed in Section XII. Furthermore, an intelligent approach to filter design permits one to pack a great deal of information in each spatial filter about true and false targets. Such filters will therefore be quite robust with respect to target perturbations while discriminating well against false targets. Techniques to do this are discussed in the next few sections.

Phase-only filters are not problem free. Some serious problems related to target translation with respect to a background and changes in the target/background contrast are discussed in Section XIV and Section XV. However, most of these problems can tentatively be solved by the techniques discussed in Section XVI.

VII THE SIGNAL-TO-CLUTTER RATIO OF A SPATIAL FILTER

The standard correlation setup in the plane $\mathbf{R}^2$ using discriminant functions will be sketched. As discussed in Section V, $\mathbf{R}^2$ is approximated in regions near the origin (0,0) by a discrete torus. It is on such a discrete torus that spatial filter design and computer simulation takes place. This discussion can be carried out in an analogous fashion for such a discrete torus. While this discussion will take place in the context of spatial filter design for implementation in an optical correlator, such filters work equally well if not better in a digital correlator, for that is exactly what is done in the simulations discussed in this report. References 3 — 10 contain exhaustive treatments of the background on optical correlation necessary to understand this section.

For any pair of complex-valued square-integrable functions f and g defined on the plane, define the symbol $\langle f,g \rangle$ to be the usual complex inner product $\int_{\mathbf{R}^2} f(y)g^*(y)dy$. Here $y = (y_1,y_2)$ is a generic point in the plane, and g^* is the complex conjugate of g. The symbol $\langle f,g \rangle$ is linear in f, conjugate linear in g, and $\langle f,g \rangle = \langle g,f \rangle^*$. Note that $\langle f,f \rangle$ is the square of the usual L^2-norm of f. For any pair of points $x = (x_1,x_2)$ and $y = (y_1,y_2)$ in the plane and any function g on the plane, let $g_x(y) = g(y\text{-}x)$. If g is initially viewed as being centered over the origin (0,0), then g_x should be viewed as being g translated so as to be centered over x. Any complex-valued square-integrable function f on the plane may be considered to be an image. This level of generality will prove useful in later sections. A good discriminant function h is a complex-valued square-integrable function on the plane which has the property that the magnitude of $\langle f,h_x \rangle$ or equivalently $| \langle f,h \rangle |^2$ is relatively large if there is an object of interest in f near x and has the property that the magnitude of $\langle f,h_x \rangle$ (or equivalently $| \langle f,h \rangle |^2$) is relatively small otherwise. Recall that $\mathcal{F}$ denotes the two-dimensional Fourier transform operator. It is a simple consequence of the Parseval theorem that $\langle f,h_x \rangle = \mathcal{F}^{-1}(\mathcal{F}(f)\cdot\mathcal{F}(h)^*)(x)$. This formula indicates the manner in which a discriminant function can be implemented in an optical correlator of standard design. This implementation is summarized in the following algorithm:

(1) input the coherent image f;

(2) use a lens to compute the Fourier transform $\mathcal{F}(f)$;

(3) multiply $\mathcal{F}(f)$ (using a transparency, spatial light modulator, etc.) by the filter $\mathcal{F}(h)^*$ corresponding to the discriminant function h;

(4) use a lens to compute the inverse Fourier transform $\mathcal{F}^{-1}(\mathcal{F}(f)\cdot\mathcal{F}(h)^*)$ of this product;

(5) use a detector to measure the light intensities $|\langle f,h\rangle|^2$.

A particular choice of device used to carry out the crucial multiplication of $\mathcal{F}(f)$ by the filter $\mathcal{F}(h)^*$ in step (3) imposes severe constraints on admissible choices of the filter $\mathcal{F}(h)^*$. These constraints must be taken into account in filter design.

In a properly designed optical correlator the pixels on the detector should be in a one-to-one correspondence (up to scaling) with the same set of pixels in the input plane. Input devices and detectors are, of course, bounded in extent. Therefore all images encountered in practice of necessity must vanish outside the bounded region in the plane corresponding to the face of the input device. This basic fact must be taken into consideration in filter design. It may be assumed that the pixels on the detector correspond to a subset, perhaps all, of the pixels on the input device. There is no point in making the face of the detector any larger. Having chosen a carefully crafted filter $\mathcal{F}(h)^*$, there then will be something of interest in the image f at those places x where $|\langle f,h\rangle|^2$ is relatively large and nothing of interest at those places x where $|\langle f,h\rangle|^2$is relatively small.

A mathematical formulation of the intrinsic performance or signal-to-clutter ratio of a discriminant function h, implemented in an optical correlator of standard design, against a training set of images will now be given. This mathematical formulation will be nothing but a numerical measure of the most straightforward and naive properties that a good discriminant function should possess. We now proceed as we would in analyzing our first high school physics word problem. Just what properties do we desire for our spatial filter $\mathcal{F}(h)^*$. We proceed to first put these desires into words. When one of the centered true target images $f_1,\ldots,f_n$ is input to the correlator, we desire that there be a strong sharp light intensity near the center of the correlation plane. Furthermore, we would like the smallest of these strong recognition signals near the center of the correlation plane to be as large as possible. Next, when one of the false target images $f_{n+1},\ldots,f_m$ is input to the correlator, we want all of the light intensities in the correlation plane to be small. In fact, we would like the largest of the signals in the correlation plane to be as small as possible. These two

desires are somewhat opposed to one another. A typical reasonable compromise is to divide the quantity we want to maximize by the quantify we want to minimize.

Next, just as in analyzing our first high school physics word problem, we will quantify our analysis of the problem. Let $f_1,\ldots,f_n,\ldots,f_m$, the training set, be a finite sequence of images. The images can be ordered so that $f_1,\ldots,f_n$ will be true targets. In this discussion true targets in the training set are always centered over the origin (0,0) in the input plane. The other training set images $f_{n+1},\ldots,f_m$ will be false targets. In this formulation there need not be any false targets but there must always be at least one true target. Note that n will equal m if there are no false targets. For each integer i, $1 \leq i \leq n$, let B_i be a box or any other region in the correlation plane. Each $B_i(1 \leq i \leq n)$ certainly should be a subset of the detector face close to the origin (0,0) — indeed, a typical choice for $B_i(1 \leq i \leq n)$ is the the origin $\{(0,0)\}$ in the correlation plane. In order for h to be a good discriminant function the largest of the measured optical intensities $|\langle f,h\rangle|^2(x \text{ in } B_i)$ in the correlation plane should be relatively large for each integer i between 1 and n. Here the best choice of x in B_i may well vary with the choice of the integer i between 1 and n. In addition, the smallest of these relatively large signals should be as large as possible. In addition, let B_i be empty ($n+1 \leq i \leq m$) for convenience. Finally, let R_i be a region in the correlation plane which is contained in the complement of the $B_i(1 \leq i \leq n)$. Typical choices for the R_i are $R_i = \phi$ for $1 \leq i \leq n$ and R_i is the entire correlation plane for $n+1 \leq i \leq m$. The intensities in each of the R_i are to be regarded as false signals and we wish to suppress them as much as possible. Note that even though the $f_i(1 \leq i \leq n)$ are true targets, we may wish to suppress secondary light intensities in parts of their correlation planes. Therefore it is plausible that the $R_i(1 \leq i \leq n)$ need not be empty. Hence we desire that each of the false signals $|\langle f,h\rangle|^2(1 \leq i \leq m)$ should be small for all choices of x in $R_i(1 \leq i \leq m)$. The first, most basic fundamental figure of merit used in this work is the signal-to-clutter ratio of the discriminant function h, defined to be

$$SCR(h)=T(h)/C(h) \tag{1}$$

where

$$T(h)= \min_{1\le i\le n} \max_{x\in B_i} |<f_i,h_x>|^2 \tag{2}$$

and

$$C(h)= \max_{1\le i\le m} \max_{x\in R_i} |<f_i,h_x>|^2 \tag{3}$$

If there are no nonempty R_i, set C(h) to be one. In this case SNR(h) degenerates to be T(h). Call T(h) the threshold of h and call C(h) the clutter of h. T(h) is the largest threshold which will accept each of the images $f_{1...},f_n$ as a true target. This definition makes sense, for it is merely the ratio of the smallest peak signal given by h off a true target to the largest false signal given by h off any of the targets. SCR(h) will sometimes be called the signal-to-clutter ratio of the spatial filter $\mathcal{F}(h)^*$. Since there is a one-to-one correspondence between the discriminant function h and the spatial filter $\mathcal{F}(h)^*$, no confusion will result from such terminology.

SCR is a slight generalization and reworking of the definition of signal-to-noise ratio introduced in Kallman, Reference 11. In the case n = m = 1, SCR coincides with the notion of signal-to-clutter ratio used in radar (Skolnik, Reference 1). For any given discriminant function h, the number SCR(h) depends on the choice of f_i, B_i, and R_i, but this dependence is usually not explicitly noted when it is clear from context.

The use of SCR(h) as a measure of merit for spatial filters has been criticized on occasion in published papers as being unmotivated. An example of such criticism is examined in detail in the next section. The author believes that the discussion given in previous paragraphs makes it clear that the definition of SCR given here is precisely that which would be used by any thinking high school physics student.

Various caviling complaints are made in Kumar, Reference 12, about the lack of noise considerations if SCR(h) is used as the measure of the performance of the spatial filter $\mathcal{F}(h)^*$. These quibbling complaints might be apropos if one could accurately determine what sort of noise is involved in degrading filter performance, how the noise is intertwined with the true signal, and whether it is a significant effect compared to the effects represented by the clutter C(h). A matter on which the author is quite confident is that the bad correlation

plane effects produced by even very large amounts of additive zero mean noise on the input image are very, very minor compared with those correlation plane effects produced by not suppressing C(h). This has been gleaned from hundreds of test cases, samples of which are routinely discussed in the author's various papers. For example, two of these test cases are discussed in Kallman, Reference 13. Other examples are given in Section XVI. It is the author's opinion that the numerous papers devoted to suppressing zero mean additive noise effects in spatial filter design are sterile academic exercises which miss the point of the true difficulties in spatial filter design.

VIII FORMULATION OF SPATIAL FILTER DESIGN AS A MATHEMATICAL OPTIMIZATION QUESTION

We continue in the same spirit in this section as we did in the previous Section VII. That is, we continue as if we were analyzing our first high school physics word problem.

In Section VII we devised a mathematical function SCR(h), the signal-to-clutter ratio, as a measure of performance of the spatial filter $\mathcal{F}(h)^*$. Just as in a high school physics word problem, the next step is to identify all the free parameters involved in the definition of SCR(h). There are certain free parameters which involve the physical features of the optical correlator. Among these are the number of pixels on and/or the resolution of the input device; the number of pixels on and/or the resolution of the detector; the physical dimensions of the correlator; the characteristics of the lenses; the wavelength of coherent radiation used; and the proper choice of scaling in the Fourier or filter plane (i.e., what portion of the the filter plane should be intercepted or filtered by the spatial light modulator?) These are important engineering questions that probably should be resolved by device constraints and engineering tradeoffs determined by the particular application of the correlator. More likely the issue of the choice for these gross parameters will be/has been settled in some ad hoc manner by the manufacturers of correlators. The proper choice of these free parameters is not addressed here. The choice of the f_i and the corresponding R_i and $B_i (1 \leq i \leq m)$ can only be determined by an engineering methodology involving much experimentation and computer simulation studies. The f_i used in this effort were discussed in Section III. The B_i, after some experimentation, were usually chosen to be just the origin $\{(0,0)\}$ in the correlation plane. The R_i usually were chosen to be equal to $\emptyset$ for $1 \leq i \leq m$ and were usually taken to be the entire correlation plane for $n+1 \leq i \leq m$. The remaining free parameters are entries in the matrix representing the filter $\mathcal{F}(h)^*$, each of which is represented as an independent complex number, or equivalently is represented as a function of one nonnegative parameter and one real parameter, the amplitude and the phase of the entry. Equipment constraints, light budget constraints, storage requirement constraints, and other considerations can put limitations on the range of some of these parameters.

It is convenient on occasion to have the option of permanently fixing some of the entries

of the spatial filter $\mathcal{F}(h)^*$ to be zero. The complement of the set of entries of $\mathcal{F}(h)^*$ which are permanently set to be zero will be called the region of support of $\mathcal{F}(h)^*$. This should be done by setting up an array of booleans representing a filter boolean mask, stored in a computer file, say fbm.dat, whose entries are in a one-to-one correspondence with the entries of $\mathcal{F}(h)^*$. The entries of fbm.dat are either TRUE or FALSE: FALSE if the corresponding entry in $\mathcal{F}(h)^*$ is fixed to be 0, TRUE otherwise.

A general spatial filter can be considered to be a square L×L array $[A_{ij}z_{ij}]$, where the $A_{ij} \geq 0$ represent amplitudes and the z_{ij}, complex numbers of modulus one, represent phases. The z_{ij} are uniquely determined if $A_{ij} \neq 0$ and are permanently fixed to be one if A_{ij} is permanently fixed to be zero. Any possible addressable device which could implement a general spatial filter would of necessity have a finite dynamic range. We must therefore make some assumption limiting the range of the amplitudes. We can assume that each A_{ij} satisfies $0 \leq A_{ij} \leq 1$ by multiplying the amplitudes by a correct choice of positive scaling factor if necessary. This scaling factor can be chosen so that $\max_{1\leq i,j\leq L} A_{ij}=1$ if the filter is not identically zero. We will assume that the A_{ij} are constrained so that this equality always holds. Any intelligent high school physics student would now approach the construction of a general spatial filter by choosing a discriminant function h so that if $\mathcal{F}(h)^* = [A_{ij}zij]$, $1 \leq i,j \leq L$, then SCR(h) is optimized over those A_{ij}, z_{ij} for which $A_{ij} \neq 0$, with A_{ij} constrained such that $0 \leq A_{ij} \leq 1$ and $\max_{1\leq i,j\leq L} A_{ij}=1$ and with the corresponding z_{ij} constrained to be a complex number with $|z_{ij}| = 1$. The general spatial filter $\mathcal{F}(h)^*$ which results from the solution of this constrained optimization question would hopefully be a good general spatial filter for the training set $f_1,\ldots,f_n, f_{n+1},\ldots,f_m$.

Spatial filters for which the A_{ij} are constrained to be zero or one are of special interest. These are spatial filters whose entries are zero or an arbitrary continuous phase. As a concession to those (and there are many) who equate progress with the creation of acronyms, we will call such spatial filters zero and phase (ZAP) filters. Therefore each ZAP filter is an L×L array of the form $[A_{ij}z_{ij}]$, where A_{ij} is either zero or one and z_{ij} is a complex number of modulus one. A_{ij} is zero if and only if the (i,j)-th entry of fbm.dat is FALSE and A_{ij} is 1 if and only if the (i,j)-th entry of the boolean mask fbm.dat is TRUE. If A_{ij} is zero, then we fix z_{ij} to be one. The free parameters in a ZAP filter are precisely those complex

numbers of modulus one (or phases) z_{ij} for which A_{ij} is one. The ZAP filter $\mathcal{F}(h)^*$ should be designed by varying these z_{ij} in an intelligent manner so as to drive SCR(h) to as high a level as possible.

The author has invented algorithms to solve these optimization questions and has implemented them in occam 2 codes for use on transputer parallel processing systems. These algorithms are discussed in detail in the next two sections. They were first discussed in the literature by the author in References 14 — 15. They are very efficient iterative techniques. The very first initial guess needed by these iterative optimization codes can be generated by elementary linear combination techniques. These issues are discussed in more detail in Appendices A-G. Appendix B is a good overview of the software created by the author. The practicality of these algorithms is discussed in Appendix J.

The author has been using this approach to spatial filter design since 1986. Other than the author, virtually all approaches to general spatial filter design are of the following form. Given a training set consisting of true targets $f_1,\ldots,f_n$ and false targets $f_{n+1},\ldots,f_m$, and some objective function φ which purports to measure filter performance, find that linear combination of the training set images which maximizes φ. This approach therefore uses only m free parameters to design a general spatial filter — this in a problem where there are naturally $2L^2$ free parameters. See Kumar, Reference 12, for a rather broad survey of such techniques and for further references. It is the author's experience that these techniques produce useful filters in cases for which there are few if any false targets and where the true target set consists of images which are relatively small perturbations of one fixed target. However, results appear to be quite poor in cases where there are many dissimilar true targets and/or even a few nontrivial false targets. Such results reinforced the author's intuitive belief that good general spatial filter design is not an exercise in linear algebra but is an optimization question of extreme complexity involving a great many free variables. In this context the linear combination techniques for spatial filter design should be regarded as the first two terms in the Taylor expansion for what one really desires. In general one should not expect good results from such an approximation anymore than one should expect to find good approximations to the function exp(x) by studying the function $1 + x$

The author's approach to spatial filter design has been criticized by Jared, Reference

16, since it allegedly "yields little information about the theory of distortion invariant filter construction." The author does not understand this objection. Even the simplest calculus questions or linear programming questions lead to solutions for which the best choice of free parameters does not have an obvious or pleasant physical interpretation. Why should we expect a simple physical interpretation for the optimal choice of free parameters in this extremely complicated problem which might typically have over 32,000 free parameters? Jared's own approach to general spatial filter design, like that of so many others, is a linear combination approach. Perhaps he can inform us whether or not his approach to general spatial filter design leads to great physical insights. Perhaps he can also inform us why we should expect his general spatial filters to perform optimally when only m (the size of the training set) free parameters are used in his design process — this in a problem which typically has over 32,000 free parameters.

IX THE CHOICE OF AN OPTIMAL SEARCH DIRECTION IF THE FREE PARAMETERS ARE UNCONSTRAINED

This section is a rewrite and extension of a topic previously discussed in References 17 — 18. The ideas presented here were essentially discovered by the author in 1985. The contents of this section certainly must be well known, but it is difficult to give a reference for a coherent discussion of the topics considered here. The methods described here are apropos to the design of ZAP filters as formulated in Section VIII. These techniques are not directly useful for the design of general spatial filters as formulated in Section VIII, for this is a constrained optimization problem and is much more difficult. Techniques and algorithms to solve the constrained optimization question involved in the design of general spatial filters are discussed in the next section. However, some of the techniques developed in this section will prove useful as an intermediate step in the constrained optimization problem.

Optimization questions of various sorts are familiar to anyone who has taken a calculus course or an operations research course. The sort of optimization question of interest in this section does not seem to fall neatly into either of these subject areas, but rather involves aspects of both. The ZAP filter design problem discussed in Section VIII is a special case of the following general optimization question. A more detailed discussion of this special but vitally interesting case will be given at the end of the following general discussion.

Let X be a nonempty open subset of the k-dimensional Euclidean space $\mathbf{R}^k$. Let $\varphi_1, ...,$ φ_N be a finite sequence of differentiable functions on X. Define a function φ on X by setting

$$\varphi(\mathrm{x}) = \min_{1 \le i \le N} \varphi_i(\mathrm{x}) \tag{4}$$

for each x in X. Problem: Find an iterative algorithm to drive φ to a local maximum on X. This problem is somewhat involved, for even though each φ_i may be as differentiable as could be desired, φ itself almost always is not. This is not merely a pedantic point, for all of the difficulty in optimizing φ occurs at those points where φ is not differentiable. Hence, simple gradient steepest ascent methods do not apply here. In the ZAP filter design problem considered in Section VIII, φ is actually a function on a very large torus or product of circles and not an open subset of some Euclidean space. However, any such torus is

locally parameterized by an open subset of some Euclidean space, and the discussion here encompasses the problems considered in previous sections. In the previous sections a typical size for k is 16,384 and N can vary between the relatively small number of images in the training set when optimizing a threshold to a number in the millions when optimizing a signal-to-clutter ratio. Because of the large number of functions involved and the large number of variables, it is important to use algorithms in which no large sets of linear equations need be solved or large matrices inverted. The algorithm sketched below has this property.

Choose an initial point x_0 in X. An iterative scheme involves the choice of a feasible direction (or unit vector) d in $\mathbf{R}^k$ so that $\varphi(x_0) < \varphi(x_0 + \alpha d)$ for some sufficiently small positive number α. Given that such a direction d exists, it is desired to choose the best possible d. If no such d exists, then the iteration stops. In order to determine whether or not such a d exists, one proceeds as follows. Let F be the set of all integers i between 1 and N such that i is active, i.e., such that $\varphi_i(x_0)$ is very close to $\varphi(x_0)$. Sometimes, particularly at the beginning of an iteration, the size of F might be 1, but in practice it quickly grows to be rather large. The size of F of course depends on what tolerance is used to determine when $\varphi_i(x_0)$ is very close to $\varphi(x_0)$. This tolerance usually should go to 0 as the iteration proceeds. Calculate the gradients $v_j = (\nabla\varphi_j)(x_0)(j \in F)$. A direction d is feasible provided each dot product $v_j \cdot d$ is positive. The best direction d to choose is the solution to

$$\max_{\substack{d \in R^k \\ |d|=1}} \min_{j \in F} v_j \cdot d \tag{5}$$

where $|x|$ is the length of the vector $x \in \mathbf{R}^k$. Apparently one intractable problem has been replaced with another. However, this is not the case, for the vector d which solves Equation (5) admits an interesting geometric interpretation.

Let P be the polytope in $\mathbf{R}^k$ spanned by the vectors v_j $(j \in F)$. P in fact is the convex hull of the vectors v_j $(j \in F)$, viz.,

$$P = [\sum_{j \in F} t_j v_j | t_j \geq 0, \sum_{j \in F} t_j = 1].$$

Note that if the origin $0 \in P$, then there is no unit vector d such that $v_j \cdot d > 0$ $(j \in F)$. For if $v_j \cdot d > 0$ $(j \in F)$, then $v \cdot d > 0$ for all $v \in P$. But $0 \cdot d = 0$. This is an obvious

contradiction, and so there is no unit vector d such that $v_j \cdot d > 0$ (j $\in$ F) if 0 $\in$ P. In this case either the tolerance used in determining F must be shrunk or the algorithm must terminate, for there is no feasible direction in which to move to try and improve φ. So it may be supposed that 0 $\notin$ P. P is a compact convex subset of $\mathbf{R}^k$ and therefore contains a unique point w $\in$ P which is closest to but distinct from **0**. Then d = w/|w| is the unique solution to Equation (5). To prove this, note that

$$|tv + (1\text{-}t)w|^2 \geq |w|^2$$

for all 0 < t <1 since P is convex and w is the point of P with smallest length. Expanding, this implies that

$$t^2 v \cdot v + 2t(1\text{-}t)v \cdot w \geq t(2\text{-}t)w \cdot w.$$

Divide this expression by 2t and then let t $\rightarrow$ 0+. One obtains that

$$v \cdot w \geq w \cdot w$$

for all v $\in$ P. This implies that

$$v \cdot d \geq w \cdot d = |w|$$

for all v $\in$ P. This in turn implies that

$$w \cdot d = |w| = \min_{j \in F} v_j \cdot d$$

since w is a convex combination of the v_j (j $\in$ F). Now let d′ be any other unit vector distinct from d = w/|w|. Then

$$\min_{j \in F} v_j \cdot d' \leq w \cdot d' < |w| \cdot |d'| = |w| = \min_{j \in F} v_j \cdot d$$

by the strict form of the Cauchy-Schwarz inequality. Hence, d = w/|w| is the unique solution to Equation (5).The above analysis implies that one can find the best direction d = w/|w| to move by finding the

$$w = \sum_{j \in F} t_j v_j$$

which solves the quadratic programming problem:

minimize $| \sum_{j \in F} t_j v_j |^2$

subject to the constraints $t_j \geq 0$, $\sum_{j \in F} t_j = 1$.

This quadratic programming problem can be solved by standard quadratic programming techniques which may be found in many texts, the classic being Dantzig, Reference 19. Efficient programming is required in solving this quadratic programming problem, for it is only an intermediate problem in each iterative step. This problem is typically solved in the author's transputer based computing system used in this study in a fraction of a second for values of k which typically can be as large as 200.

A very special, simple example or two might give the reader a bit of intuition on the origins of this quadratic programming problem. Suppose that v_1 and v_2 are two orthogonal unit vectors in the plane $\mathbf{R}^2$. It is quite intuitive that the best choice for d is the unique unit vector which makes a 45 degree angle with both v_1 and v_2. But d has another interpretation. In this case P, the convex hull of v_1 and v_2, is the line segment from v_1 to v_2. The vector closest to the origin in P is just the intersection of P with the ray through d. Thus d is as determined by the previous analysis. Again, suppose that v_1, v_2, and v_3 are three orthogonal unit vectors in $\mathbf{R}^3$. Again, it is quite intuitive that the best choice for d is the unique unit vector which makes a 45 degree angle with all three vectors. In this case P is the equilateral triangle whose vertices are v_1, v_2, and v_3 and the vector in P closest to the origin is the centroid of this equilateral triangle. But this vector points in the direction of d, as determined by the previous analysis.

We now show how the algorithm just described can be used to design a good ZAP filter $\mathcal{F}(h)^*$. We approach this problem in the most naive manner possible. First of all, for any specific $1 \leq i \leq m$ and any specific $x \in B_i$ or $x \in R_i$, $| \langle f_i, h_x \rangle |^2$ is a smooth function of the nontrivial phases in the matrix $\mathcal{F}(h)^*$. This follows easily from the Plancherel Theorem. Next, fix some suitably chosen $\epsilon > 0$ and some $\delta > 0$. Let $P(h,\epsilon)$ be the set of pairs (i,x), where $1 \leq i \leq n$, and $x \in B_i$ so that $T(h) \leq | \langle f_i, h_x \rangle |^2 \leq T(h) + \epsilon$, i.e., so that $| \langle f_i, h_x \rangle |^2$ is close to T(h). Let $Q(h,\delta)$ be the set of pairs (i,x), where $1 \leq i \leq m$ and $x \in R_i$, so that $C(h) - \delta \leq | \langle f_i, h_x \rangle |^2 \leq C(h)$, i.e., so that $| \langle f_i, h_x \rangle |^2$ is close to C(h). Let the φ then be the set of all quotients of the form $| \langle f_i, h_x \rangle |^2 / | \langle f_i, h_y \rangle |^2$, where $(i,x) \in P(h,\epsilon)$ and $(j,y) \in Q(h,\delta$. The algorithms discussed earlier in this section then can be applied iteratively to drive S(h) to an optimum. The choice for an initial choice of h is discussed in Appendix B. The choice

of $\epsilon > 0$ and $\delta > 0$ should gradually become smaller as the iterative calculation proceeds. For this algorithm to work well the B_i should be rather small subsets of the correlation plane — one should choose $B_i = \{(0,0)\}$ $(1 \leq i \leq n)$if at all possible.

X THE CHOICE OF A GOOD SEARCH DIRECTION IF SOME OF THE FREE PARAMETERS ARE CONSTRAINED

The methods described here are apropos to the design of general spatial filters as formulated in Section IX. The ideas discussed here occurred to the author in 1989. The technique described is a double iteration. The author discovered this algorithm after much experimentation to check its computational feasibility and practicality. It certainly is an open question if there are other more efficient techniques.

Let X be a nonempty open subset of $\mathbf{R}^p$ and let Y be a product of intervals in $\mathbf{R}^q$. Let $\varphi_1, ..., \varphi_N$ be a finite sequence of differentiable functions on X×Y. Define a function φ on X by setting

$$\varphi(x,y)= \min_{1\leq i\leq N} \varphi_i(x,y) \tag{6}$$

for each $(x,y) \in X\times Y$. Problem: Find an iterative algorithm to drive φ to a local maximum on X×Y. This optimization problem is much more complicated than that considered in Section IX since the y parameters must lie in the product of intervals Y rather than an open subset of $\mathbf{R}^p$. In the general spatial filter design problem considered in Section VIII, φ is actually a function on the product of a very large torus or product of circles with a product of unit intervals [0,1]. As before, any torus is locally parameterized by an open subset of some Euclidean space, and the discussion here encompasses the problems considered in Section VIII. In the previous sections typical sizes for p and q are 16,384, a total of 32,768, and N typically is in the millions when optimizing a signal-to-clutter ratio.

Choose an initial point $(x_0,y_0) \in X\times Y$. An iterative scheme involves the choice of a feasible direction (or unit vector) $(c,d) \in \mathbf{R}^p \times \mathbf{R}^q$ so that $\varphi((x_0,y_0)) < \varphi((x_0,y_0) + \alpha(c,d))$ for some sufficiently small positive number α and so that $y_0 + \alpha d \in Y$. Since X is open, $y_0 + \alpha c \in X$ for all sufficiently small α. A choice for finding such a (c,d) is the following double iteration. Fix $\alpha > 0$, usually taken to be quite small. First, ignoring the constraints for the moment, use the iterative algorithm described in Section IX to choose a feasible direction (c,d). We are done if $y_0 + \alpha d \in Y$. If so, stop the double iteration and begin afresh. If $y_0 + \alpha d \notin Y$, freeze the y-variables for which we do not have containment. Regard φ as a

function of x and the remaining y variables and iterate this procedure.

The algorithm just described can be used to design a good general spatial filter $\mathcal{F}(h)^*$. This proceeds virtually word-for-word as was described for ZAP filters in Section IX. The algorithms discussed earlier in this section then can be applied iteratively to drive SCR(h) to an optimum. The choice for an initial choice of h is discussed in Appendix B.

XI AN ALGORITHM TO CHOOSE THE REGION OF SUPPORT IN A ZAP FILTER

An algorithm to choose region of support in a ZAP filter is presented. This algorithm involves a simple thresholding technique applied to the general spatial filter described in previous sections. The author first discussed this topic in Reference 20, where numerous examples are given. Having determined the region of support in a ZAP filter, one can then view the remaining phases as free parameters and design a good ZAP filter by optimizing the filter's SCR using the algorithms described in Section IX.

Many people, including even the author (Reference 15), realized early on the benefits for target discrimination and clutter rejection of setting equal to zero some low frequency pixels in what would otherwise be a pure phase spatial filter. However, D. Flannery et al. (Reference 21) seem to have been the first to suggest extending this idea to possibly higher frequencies and to give an algorithm rather than an ad hoc guess on how to choose such frequencies. This algorithm was given for a training set consisting of one true target and one false target and consisted of setting equal to 0 those frequencies for which the ratio of the false target energy to the true target energy exceeded some threshold. An extension of this idea to larger training sets has apparently never been written down, though one could imagine a variety of techniques to do so if the training sets are of moderate size. What is not clear to the author, however, is whether or not there is such an extrapolation which can handle cases in which there are truly large numbers of true and false targets. For example, the author has tried several such extrapolations to an instance using standard tank imagery with 91 true targets and 91 false targets — these extrapolations gave only two nonzero pixels. Furthermore, such techniques ignore other possibilities for clutter rejection and target recognition in correlation calculations involving advantageous weighted mutual cancellations or additions at various spatial frequencies. The purpose of this section is to give a completely different approach which can handle large training sets and takes both techniques into consideration.

This section is an application of the general spatial filter described in previous sections. The algorithm given here is completely simple and straightforward. Given a training set consisting of true targets $f_1,\ldots,f_n$ and false targets $f_{n+1},\ldots,f_m$, create an optimized general

spatial filter $\mathcal{F}(g)^*$ for this training set using the techniques described in previous sections. The amplitudes in $\mathcal{F}(g)^*$ are trapped between 0 and 1, and it is plausible that those spatial frequencies which have small amplitude should be set equal to 0. This can be done with a simple thresholding or histogram technique. The choice of which threshold to use cannot be determined a priori in any nontrivial situation, but can only be derived from a great deal of simulation and engineering experimentation. Having chosen which spatial frequencies to set equal to 0, the next step is to form a filter by retaining the phases of the remaining spatial frequencies, use this ZAP filter as an initial point, keeping the support of the ZAP filter fixed, and optimize the SCR of the ZAP filter over the remaining phases using the author's design techniques. This produces a ZAP filter $\mathcal{F}(h)^*$ whose SCR has been optimized.

The importance of having spatial light modulators whose cells can assume a value of zero cannot be overemphasized. Spatial light modulators which have this capability appear to have a great advantage over those which do not, at least in circumstances where discrimination is important.

The general ideas presented herein occurred to the author quite some time ago. Their utility was quickly verified in April 1992 after their implementation in a variety of computer codes.

XII AN ALGORITHM TO OPTIMALLY DISCRETIZE A ZAP FILTER

Given that one has designed a good ZAP phase-only filter, how should one go about discretizing it in a rational manner, as might be required for implementation in an actual device? An algorithm for doing this for binary phase-only filters was first given in Reference 22. That algorithm can easily be extended to the case of discrete n-state phase-only filters. It may be described geometrically as follows. Orient the circle in counter-clockwise fashion. Let z be a point on the unit circle greater than or equal to 1.0 but strictly less than $\exp(2\pi i/n)$. Moving counter-clockwise around the unit circle, let $A_k(z)$ be the arc of the unit circle greater than or equal to $\exp(2\pi ik/n)z$ but strictly less than $\exp(2\pi i(k+1)/n)z$, for each $0 \leq k < n$. Discretize $\mathcal{F}(h)^*$ into an n-state phase-only filter $\mathcal{F}(h_z)^*$ as follows. If $\mathcal{F}(h)^*[p,q]$ is the p,q-th entry of $\mathcal{F}(h)^*$ and $\mathcal{F}(h)^*[p,q] = 0$, then set $\mathcal{F}(h_z)^*[p,q] = 0$. Otherwise $\mathcal{F}(h)^*[p,q]$ is a complex number of modulus one and so lies on some unique arc $A_k(z)$. In this case let $\mathcal{F}(h_z)^*[p,q] = \exp(2\pi ik/n)$. Finally do a one-dimensional search in your favorite manner to find that $z = z_0$ so that $SCR(h_z)$ is as large as possible. This choice of z_0 then gives, at least in this sense, an optimal way to discretize $\mathcal{F}(h)^*$ into an n-state filter. A careful optimal choice of z_0 as prescribed can yield filters with more than twice the signal-to-clutter ratio than a random choice of z when the training set is large. Moreover, especially for $n = 15$, the performance of the discretized phase-only filters faithfully mirrors that of their continuous phase-only ancestors. This generalization from 2 to n phase-states has been discussed previously in References 23 and 24.

XIII INTENSITY PHASE ENCODED IMAGES

A particularly useful algorithm for encoding intensity images as phase-only images is presented in this section. This algorithm is due to Dennis H. Goldstein (DHG) and the author. DHG and the author were led to consider this intensity phase-encoding algorithm because of real and potential constraints in spatial light modulators. Continuous and discrete ZAP filters can be designed to recognize a variety of target aspects of such encoded images. Such ZAP filters have been successfully demonstrated in Essex Corporation's ImSyn optical-digital correlator using intensity phase-encoded imagery as inputs. There appears to be a distinct advantage in using such an encoding scheme for filters designed to give a fairly stable response over quite large training sets. The advantage is particularly pronounced when the phase-only filters are designed to give an optimal response over the detector in the correlation plane.

Suppose a detector registers an intensity image $[a_{pq}^2]$ $(1 \leq p,q \leq N)$ on its active pixels, where each $a_{pq} \geq 0$. In the standard optical correlator architecture and algorithm the amplitude image $[a_{pq}]$ is then used as the input to the correlator. There is no mathematical reason why $[a_{pq}]$ should be used as the input to the correlator. It might be advantageous to input some other image $[b_{pq}]$ which is a function of $[a_{pq}^2]$ to the correlator, so long as there is a one-to-one correspondence $[a_{pq}^2] \rightarrow [b_{pq}]$ between the detected image and the inputs. After all, there is a definite but subtle difference between the actual measured pixel values a_{pq}^2 and the information content they convey. If the measured intensity image $[a_{pq}^2]$ contains an object of interest, then the correspondence $[a_{pq}^2] \rightarrow [b_{pq}]$ should commute with translations of the object of interest inside the aperture represented by the matrix of detected intensities. This can be the case in general only if there is a function $\varphi : [0, +\infty > \rightarrow \mathbf{C}$ so that $b_{pq} = \varphi(a_{pq}^2)$. Here $\mathbf{C}$ is the set of complex numbers. This rule $[a_{pq}^2] \rightarrow [\varphi(a_{pq}^2)]$ between images can be one-to-one in general only if φ is a one-to-one function, at least on the set of possible measured intensities a_{pq}^2. There is much empirical evidence that amplitudes do not contain much information and are quite troublesome, but that phases do contain a great deal of information, for signals in general and for Fourier transforms of imagery in particular (Oppenheim and Lim, Reference 25). This philosophy is suggestive but does not strictly apply

here, for translation does not commute with the Fourier transform operator. Proceeding in a quite simple-minded fashion, however, this philosophy suggests taking φ to be a complex exponential. Assume that an image has been digitized into eight bits per pixel (this could be any convenient number). Then the intensities are circumscribed to be the integers between 0 and 255. This suggests letting $b_{pq} = \varphi(a_{pq}^2) = \exp(a_{pq}^2 \pi i/255)$. Here a_{pq}^2 is divided by 255 and then multiplied by π so that $a_{pq}^2 \pi/255$ ranges between 0 and π. This is done so that $a_{pq}^2 \rightarrow \varphi(a_{pq}^2)$ is one-to-one and so that $\varphi(0)$ and $\varphi(255)$ are as far apart as possible. It is easy to think of other functions φ which have similar properties. Note that in the case at hand $\varphi(a_{ij}^2)$ always will be a complex number of modulus one lying in the first or second quadrants of the complex plane.

What might be the a priori advantages and/or disadvantages in employing intensity phase-encoded imagery as inputs to a correlator? A possible advantage is rather subtle and is based on plausible mathematical reasoning. The most difficult aspect of phase-only filter design lies not in making filters which recognize many aspects of true targets but which discriminate effectively against selected false targets. Let f be an image and d a discriminant function, both implemented in some sort of optical correlator. The optical correlation process in effect mathematically consists of translating and centering d successively over the various points of f, taking dot products, and plotting the square magnitudes of the resulting complex number. In order to discriminate against false targets, these square magnitudes should all be suppressed as much as possible. If f is a false target, then pixels of d should be thought of as complex weights which are carefully chosen to make the numerous dot products between f and the translates of f as small as possible. Now if f is an amplitude input which has several large magnitude pixels and if no mutual cancellations involving the weights of d can be arranged, a very large number of low magnitude pixels and corresponding elements of d will be utilized in canceling their effects. On the other hand, if the pixels of f consist of complex numbers of modulus one, then no small number of pixels of f should tie up numerous other elements of f and corresponding elements of d in canceling out their bad effects. A potentially serious disadvantage with the present encoding scheme is that the simplest of all false targets, namely that target each of whose pixels is equal to 0, gets encoded as an array of 1s. This potential disadvantage can be circumvented, however, by proper filters designed

to discriminate against constant inputs as well as other false targets.

The ideas discussed in this section were first published in Reference 23, were further discussed in References 24 and 26, and were patented in Reference 27. An application of these ideas to the processing of two-dimensional full polarimetric millimeter wavelength radar imagery is given in Reference 18. These references lend extensive support to the notion that intensity phase-encoded imagery is very useful in correlation.

XIV THE BACKGROUND CONTRAST PROBLEM

In many ways phase-only filters do not act at all like matched filters. One particularly compelling instance is the following experiment. Take any reasonably sized target, denoted by f, such as a m48d25.090 (NS) of Figure 2, in a blank background. Let $\mathcal{F}(\mathrm{f})^*_m$ be the classical matched filter corresponding to f, normalized so that its maximum amplitude is one, and let $\mathcal{F}(\mathrm{f})^*_p$ be the phase-only matched filter corresponding to f. These spatial filters were discussed in Section VI. Denote by g the image created from f by embedding f into a uniform background whose intensity is equal to that of the nonzero average of f. The image g is shown in Figure 7. The top down view of the correlation plane obtained by filtering g with $\mathcal{F}(\mathrm{f})^*_m$ is illustrated in Figure 8. Even though the correlation plane recognition signals are confused, one does indeed obtain a recognition signal intensity exactly the same as that obtained by filtering f with $\mathcal{F}(\mathrm{f})^*_m$. On the other hand, if one filters g with $\mathcal{F}(\mathrm{f})^*_p$ one obtains no recognition signal at all. This is illustrated in Figure 10. Care should be exercised to perform this simulation carefully for the reasons and along the lines given in Section V. One concludes from this simple example that the performance of phase-only filters with standard amplitude inputs is strongly dependent on the target/background contrast. This problem perhaps can be solved with appropriate edge enhancement and/or binarization techniques, but there surely are many instances when such techniques do not work, e.g., a target covered with a great deal of random noise.

The author discovered this difficulty in 1987. It was written down in Reference 17, where two ex post facto rather inappropriate solutions were suggested. These solutions were further discussed in Reference 28. The problem was discussed again in Reference 29, where the solution given in Section XVI was first proposed.

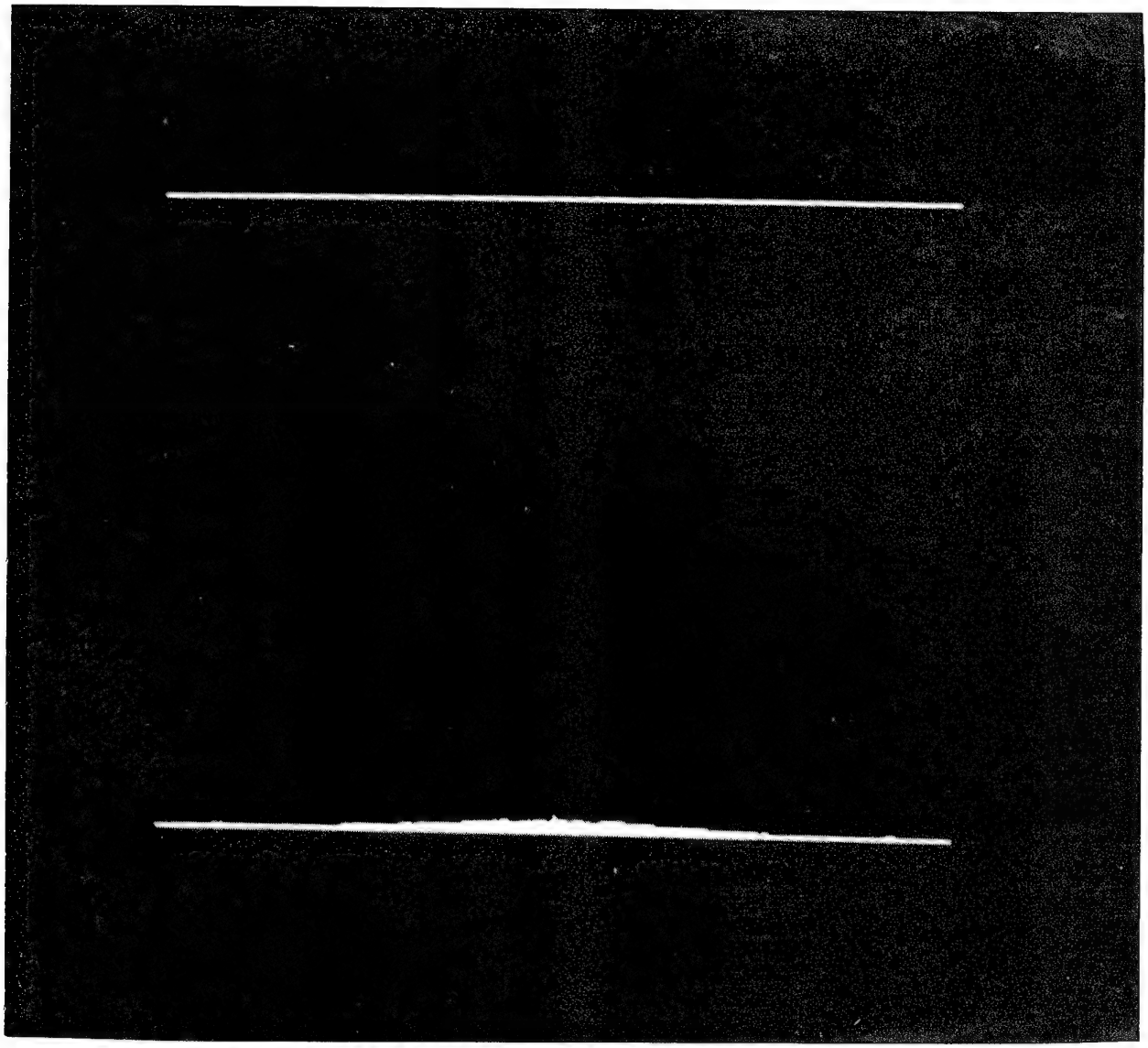


Figure 10. South-to-North View, Correlation Plane of g vs. $\mathcal{F}(\mathrm{f})_p$.

XV THE TRANSLATION INVARIANCE PROBLEM

Another basic problem for phase-only filtering arises in case the target is translated with respect to a nontrivial background. Let f and g be the tank images discussed in Section XIV. Let h be the image created by pushing f into the upper left hand corner of the array representing the input aperture and then adding the same background used to create g. If one filters h with $\mathcal{F}(\mathrm{f})^*_m$ one again gets a messy correlation plane though the peak is unaltered and correctly located. If one filters h with $\mathcal{F}(\mathrm{f})^*_p$ one again gets absolutely no response. If one creates the phase-only matched filter $\mathcal{F}(\mathrm{g})^*_p$, then of course $\mathcal{F}(\mathrm{g})^*_p$ filters g perfectly but does a much less than perfect job when filtering h. The explanation for this is that while correlation is a translation invariant process, h is not a translate of g. This lack of robustness with respect to translation with respect to a nontrivial background is even more pronounced in case this experiment is repeated with filters which involve large numbers of true targets in their training sets. Once again care must be exercised to perform this simulation carefully for the reasons and along the lines given in Section V.

The author discovered this difficulty in 1987 and first recorded it in Reference 17, where two ex post facto rather inappropriate solutions were suggested. These solutions were further discussed in Reference 28. The problem was discussed again in Reference 29, where the solution given in Section XVI was first proposed.

XVI ZERO MEAN INTENSITY PHASE-ENCODED IMAGES AS A SOLUTION

Two nontrivial difficulties in spatial filtering were discussed in Section XIV and Section XV. One potential solution to these problems is to work with edge enhanced and/or binarized imagery and to make filters from training sets of such imagery. Such techniques will certainly eliminate a uniform background. However, backgrounds are seldom uniform. Instead, to a first approximation, they consist of some sort of noise about a mean. Edge enhancements and/or binarization techniques may work for some or even most real images, but here we will concentrate on imagery for which edge enhancement and/or binarization techniques will provide little relief.

A rather heavy-handed potential solution to the problems discussed in the previous two sections was proposed in References 17 and 28. As in these references, the key philosophical point adopted here is that the differences from the average background contain all useful information for most realistic targets. To translate this philosophy into a simple algorithm, one should not filter the usual amplitude target, but instead should compute the average amplitude $\langle f \rangle$ of the target and filter on the array consisting of the amplitude target with $\langle f \rangle$ subtracted from every pixel. This concept was tried with considerable success for targets in a medium to high intensity background, but the performance of these filters faltered in low intensity backgrounds. Therefore, this algorithm needed to be modified further. An algorithm for phase-encoding intensity imagery is discussed in Section XIV. This algorithm has been shown to have several apparent advantages in correlation (References 23 and 24). It was therefore natural to study zero mean versions of target images whose intensities are properly phase-encoded. Note that since phase-encoded images are arrays of complex numbers, the mean of such an array is also a complex number. Suitably designed filters were created from training sets of such zero mean intensity phase-encoded images and their performance was gratifying in most instances. While no string of examples can prove a general statement, the empirical evidence presented in this section is typical of that observed in a large number of situations. Examples of such success are discussed in more detail in Reference 29. No correlation process using a single spatial filter can be completely invariant with respect to background fluctuations, shifts against the background, and additive zero mean

noise. However, the filters discussed here and in Reference 29 are relatively invariant with respect to such perturbations of the training set imagery. These filtering techniques might prove to be of practical importance since they can be implemented in Essex Corporation's ImSyn optical-digital processor.

The results of the following simulations partially illustrate the techniques discussed above. The true target sets were twenty-one of the m48d20 (OS) tank images at a ninety degree aspect angle plus or minus ten degrees, m48d20.080 (OS),..., m48d20.090 (OS),..., m48d20.100 (OS). The false target sets came from twenty-one of the m60d20 (OS) tank images at a ninety degree aspect angle plus or minus ten degrees, m60d20.080 (OS),..., m60d20.090 (OS),..., m60d20.100 (OS). These target sets, discussed in Section III, are rather similar. This similarity makes the recognition and discrimination problem considerably more difficult. Each M48 and each M60 was embedded into a uniform background whose pixel intensities were 128, close to the average nonzero pixel intensity level on the targets. Figure 7 shows m48d20.090 (OS) in a 128 intensity background and Figure 11 shows m60d20.090 (OS) in a 128 intensity background. These training set images were processed by intensity phase-encoding them, computing their respective complex-valued means, and subtracting their respective complex-valued means from each pixel. Filters were designed by the techniques described in previous sections and the appendices. Figure 12 is the ZAP filter with fifteen equally spaced phase-states designed for this training set of 21 true and 21 false targets. It will be denoted by $\mathcal{F}(\mathrm{h})^*_{15}$. Figure 13 is the boolean mask for $\mathcal{F}(\mathrm{h})^*_{15}$ — black represents 0 and white represents complex numbers of modulus 1.

The following simulations were done carefully as prescribed in Section V. Figures 14 — 25 illustrate a few correlation results. The subtitles are rather self explanatory. For example, Figure 14 represents m48d20.090 (OS) translated into the upper left hand corner of the input array, with the background filled in with pixels of medium intensity 128, and with mean 0.0, standard deviation 25.0 Gaussian noise added independently to every pixel. The images in a low intensity background have an average background of 50.0 and those in a high intensity background have an average background of 200.00. Exactly the same noise is added to each input image in exactly the same manner. The six south-to-north views of the correlation plane illustrate the relative immunity of the correlation process to changes in the background

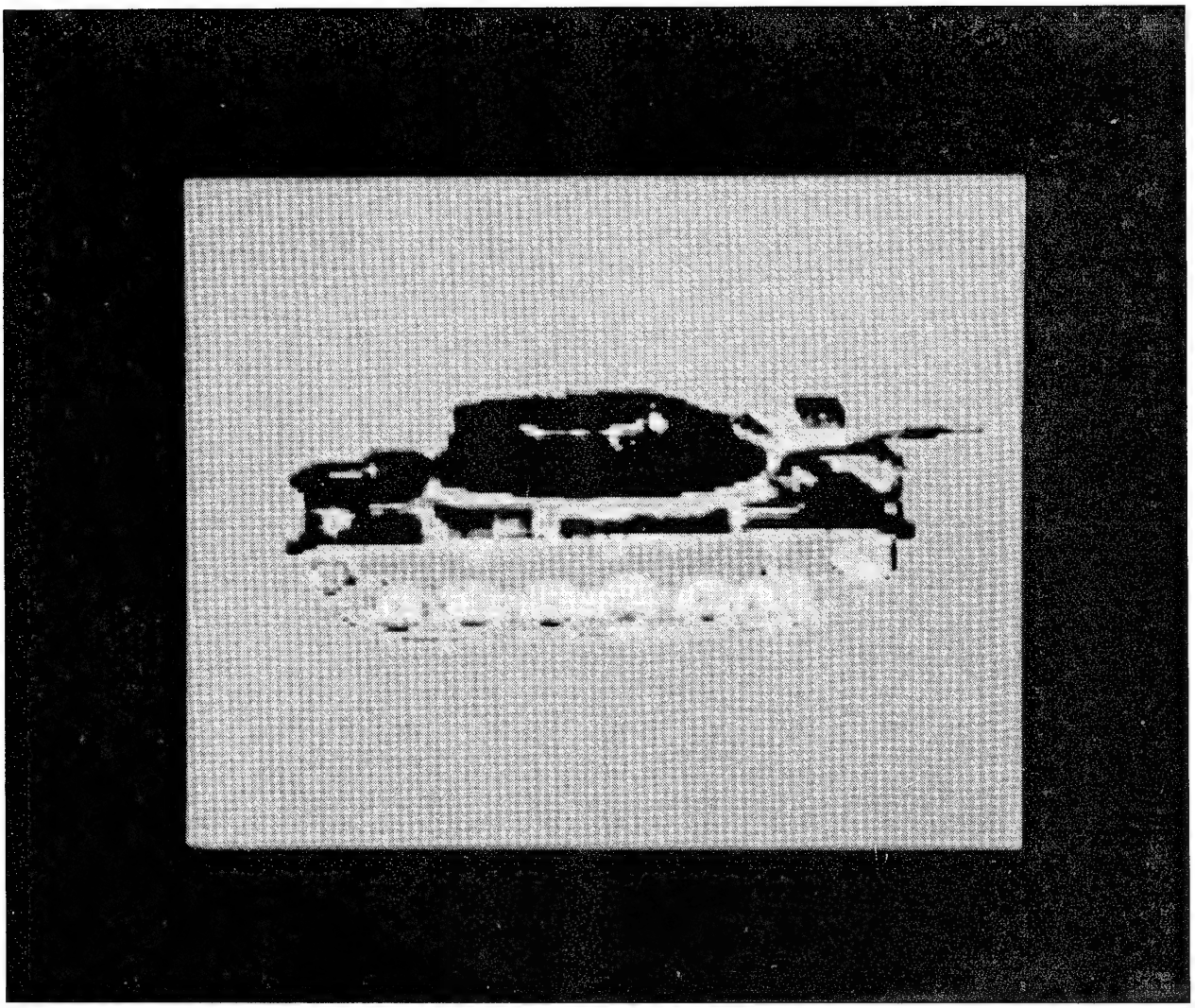


Figure 11. The M60 Image m60d20.090 (OS) in a Medium Intensity Background

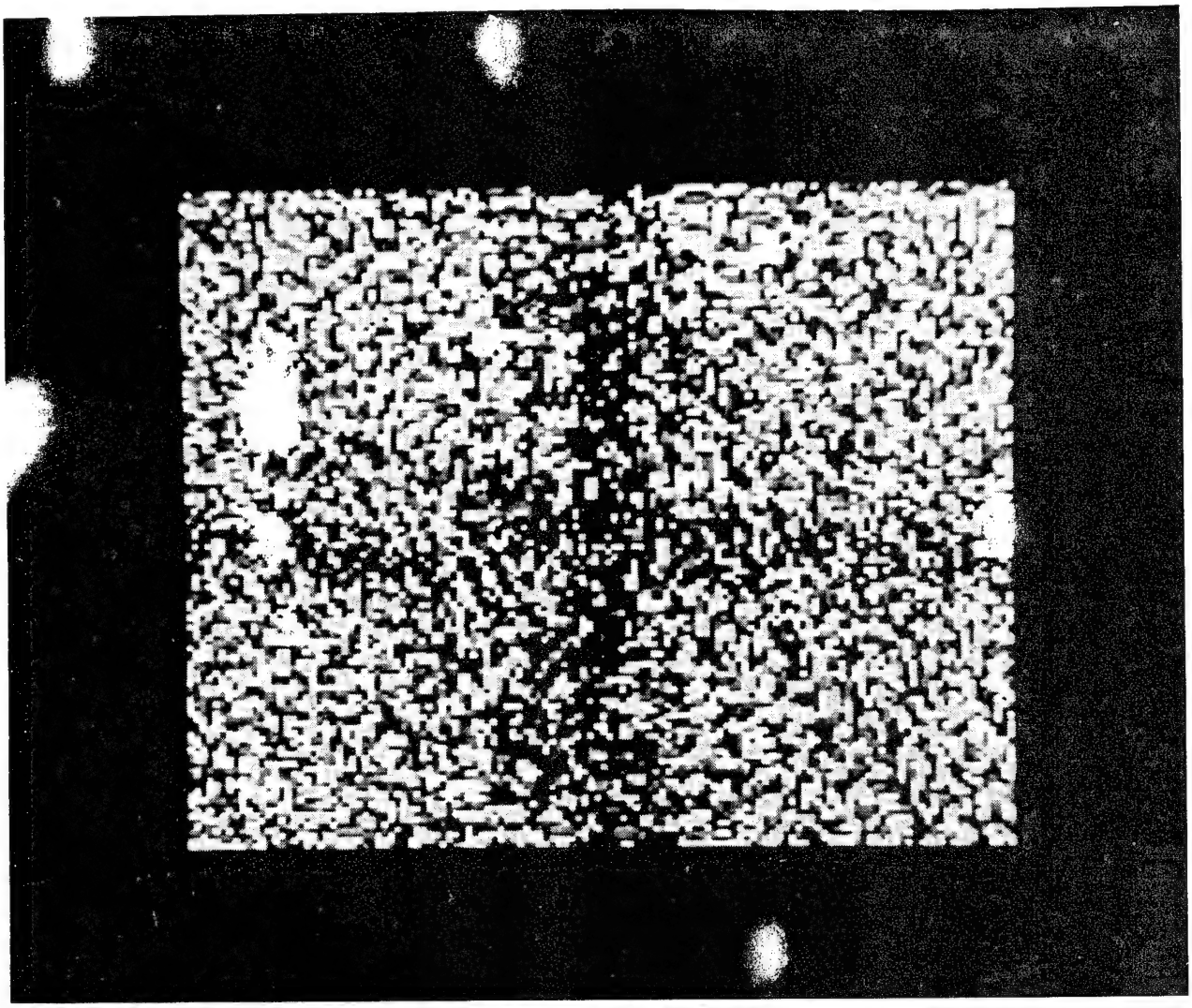


Figure 12. The ZAP Filter $\mathcal{F}(\mathrm{h})^*_{15}$

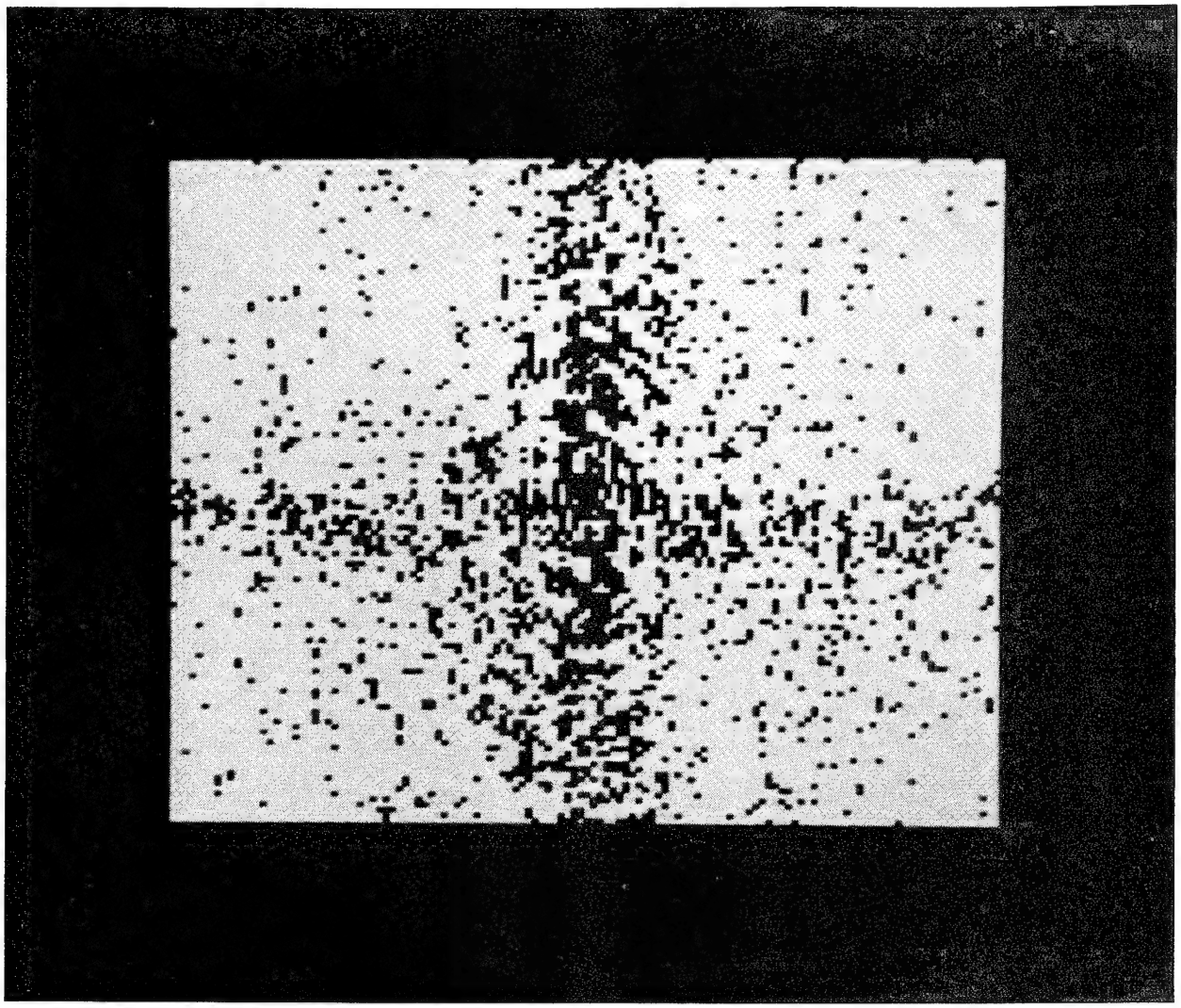


Figure 13. The Boolean Amplitude Mask of $\mathcal{F}(\mathrm{h})^*_{15}$

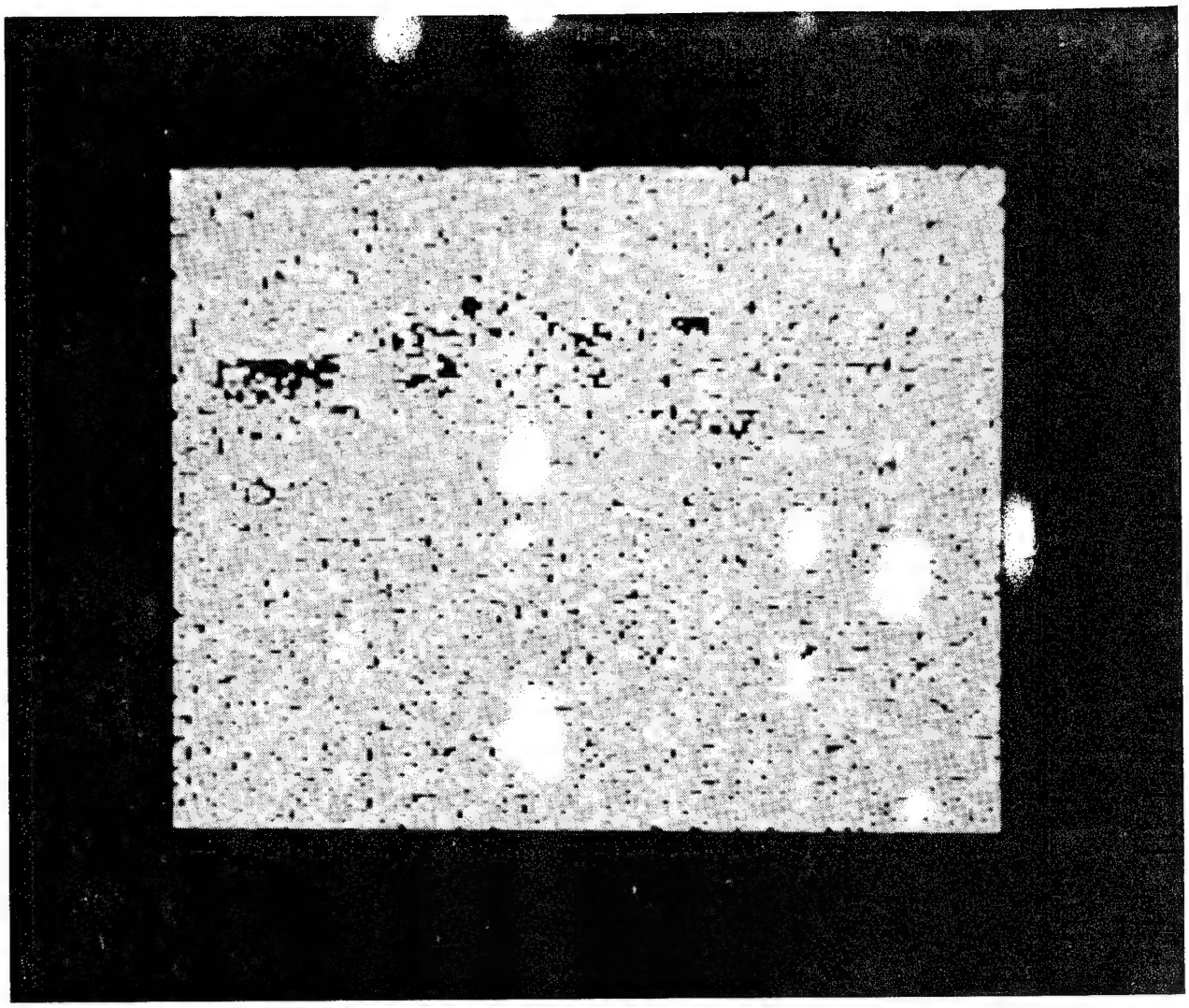


Figure 14. The Shifted and Noisy m48d20.090 (OS) in a Medium Intensity Background

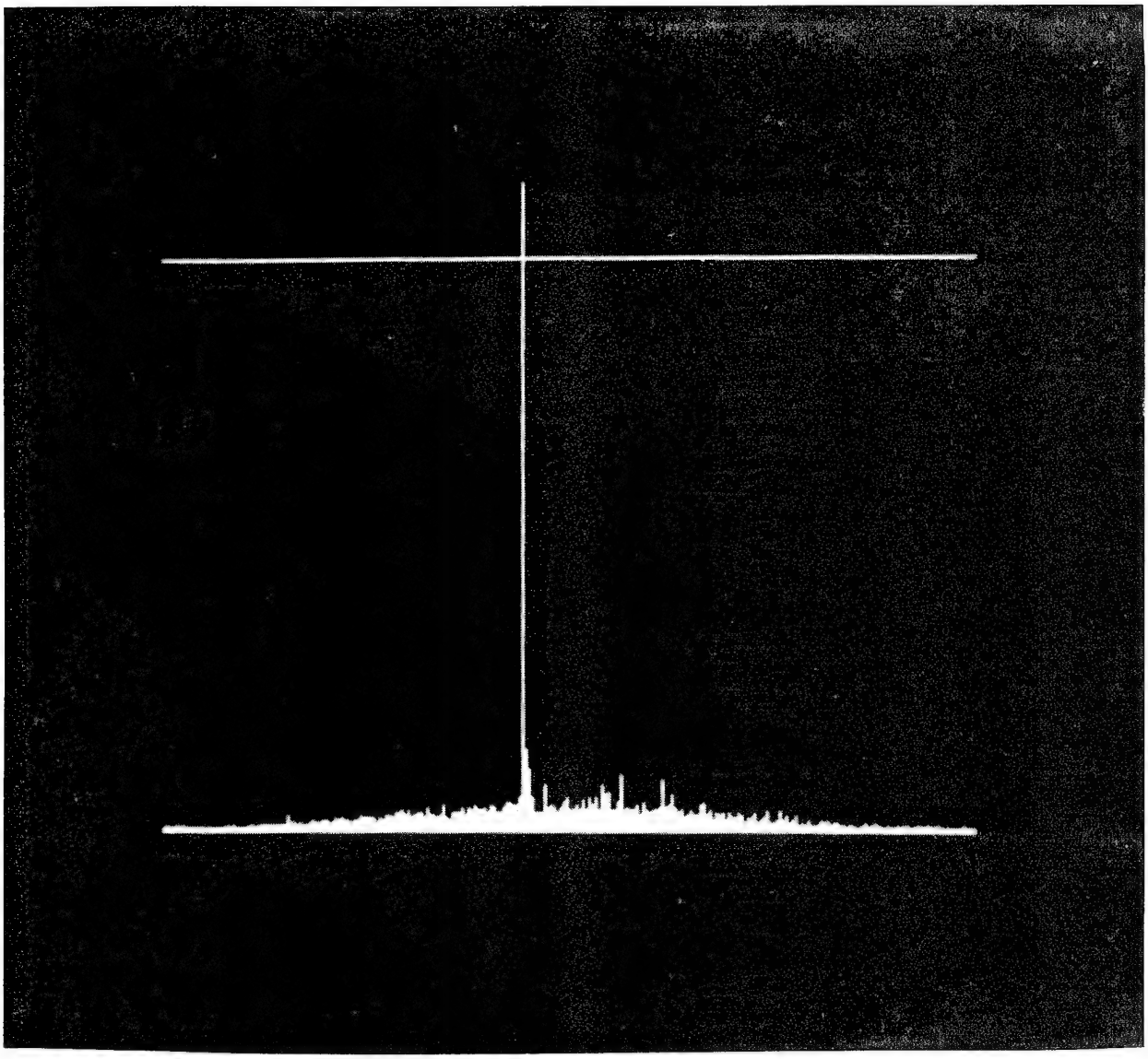


Figure 15. South-to-North Correlation Plane View, the Image of Figure 14 vs. $\mathcal{F}(\mathrm{h})^{*}_{15}$

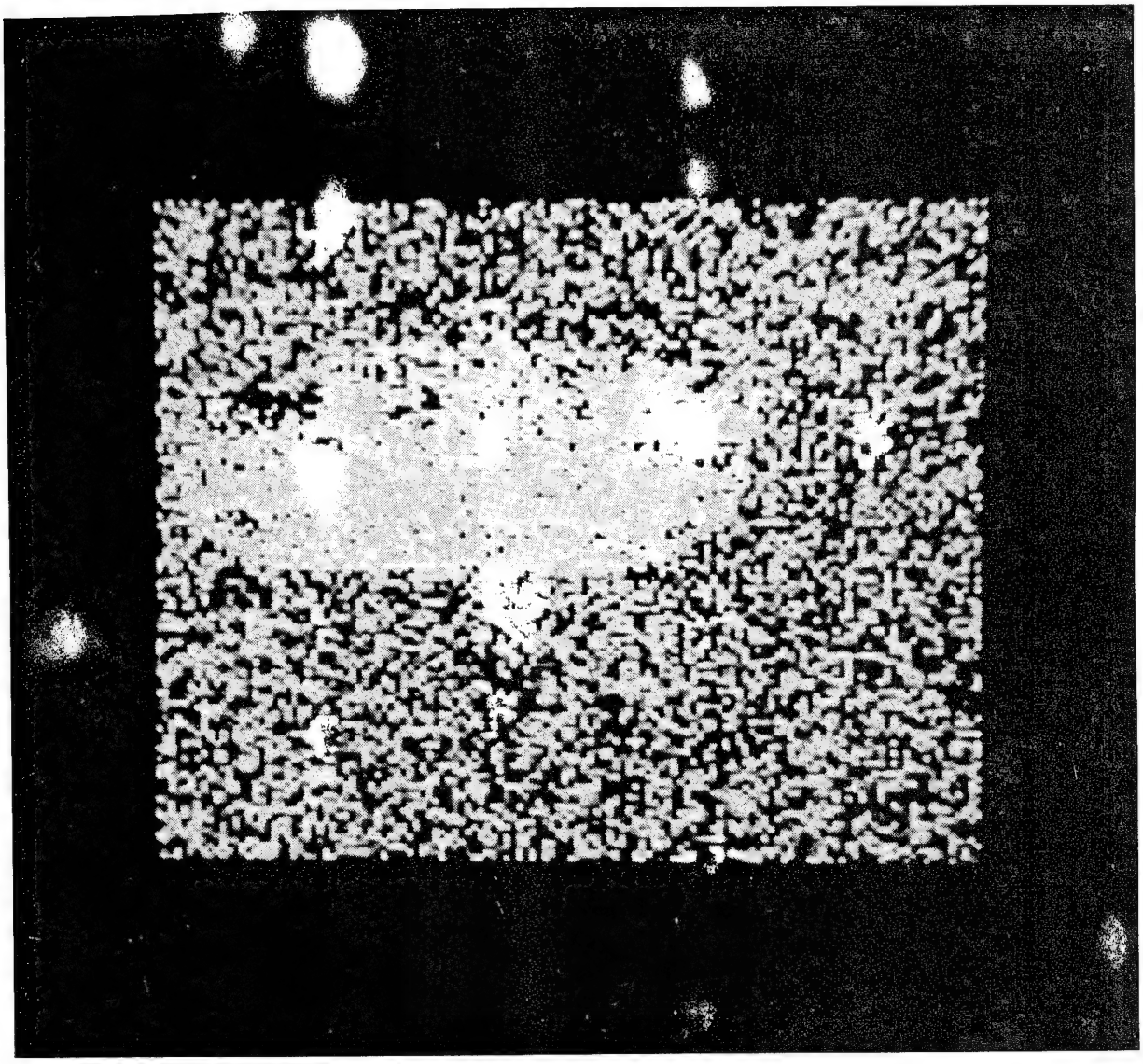


Figure 16. The Shifted and Noisy m48d20.090 (OS) in a Low Intensity Background

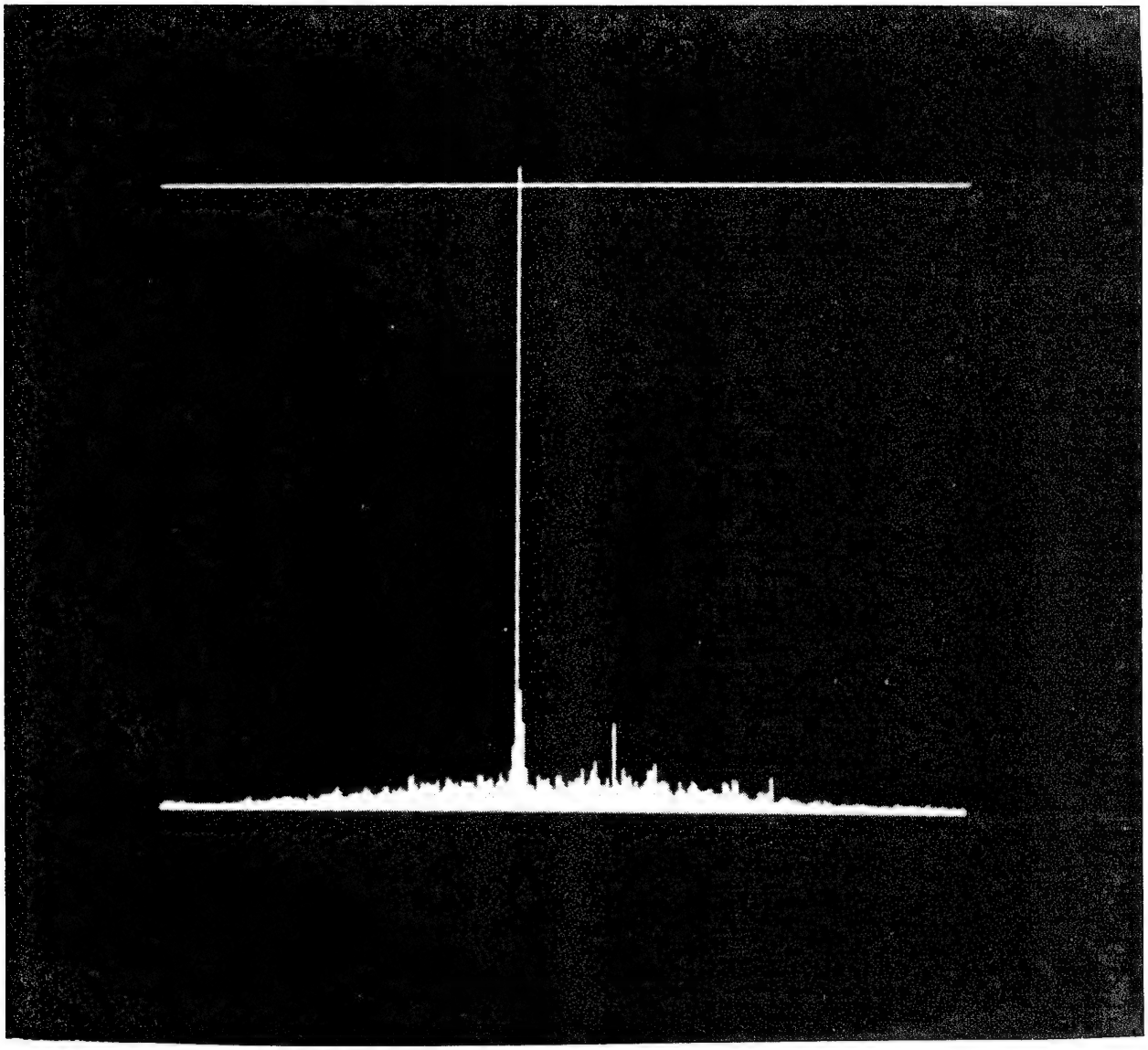


Figure 17. South-to-North Correlation Plane View, the Image of Figure 16 vs. $\mathcal{F}(\mathrm{h})^{*}_{15}$

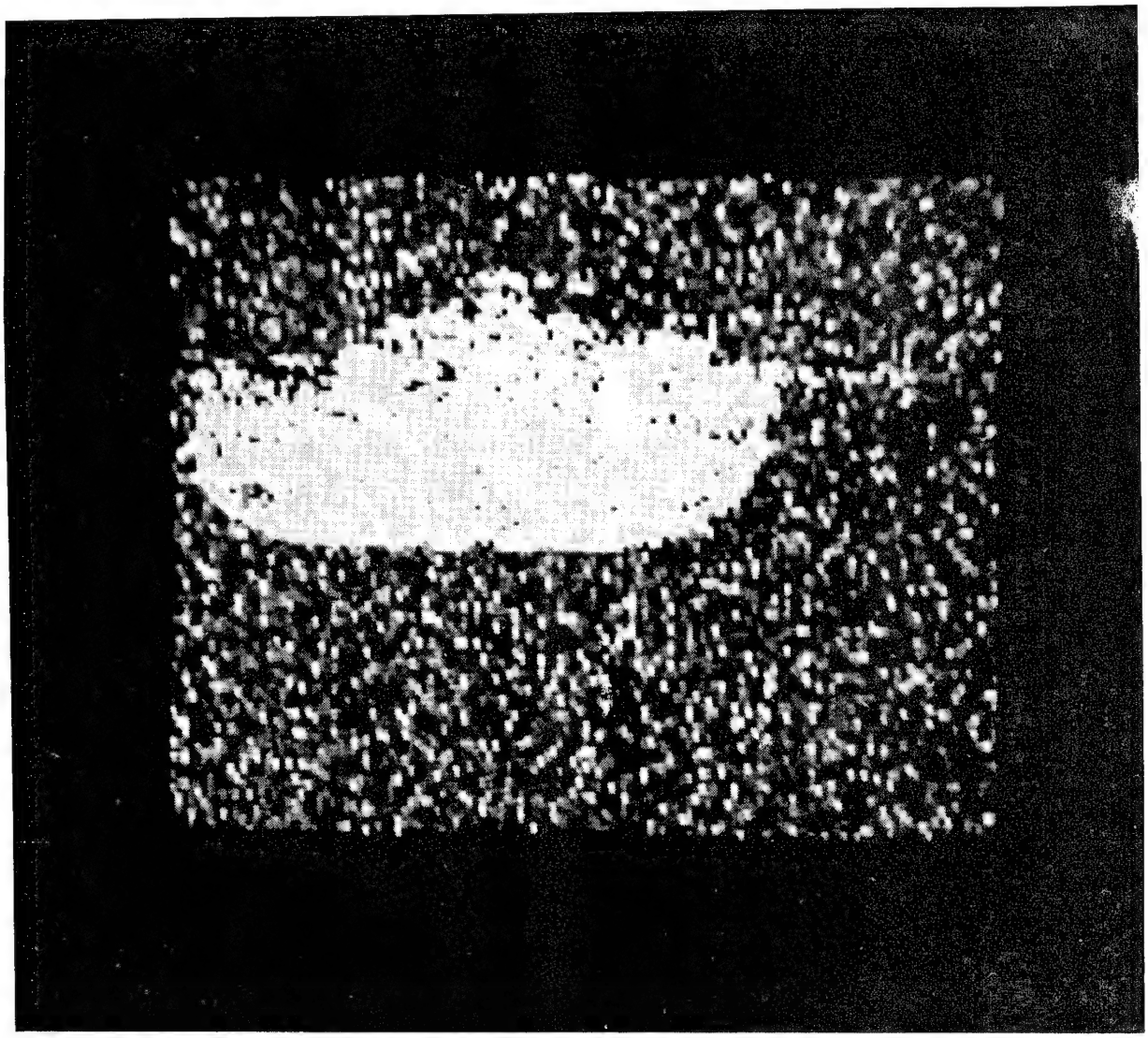


Figure 18. The Shifted and Noisy m48d20.090 (OS) in a High Intensity Background

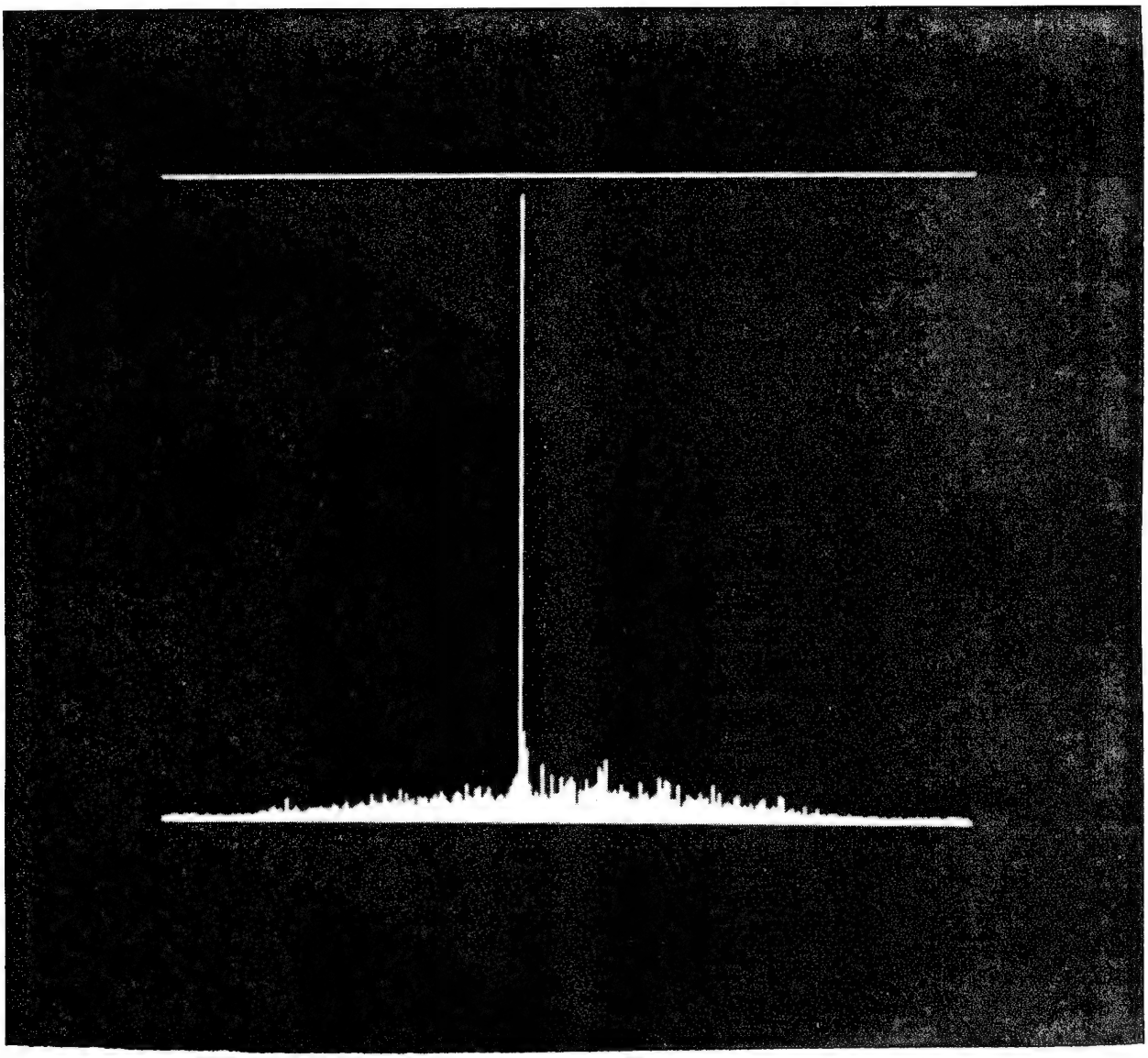


Figure 19. South-to-North Correlation Plane View, the Image of Figure 18 vs. $\mathcal{F}(\mathrm{h})^{*}_{15}$

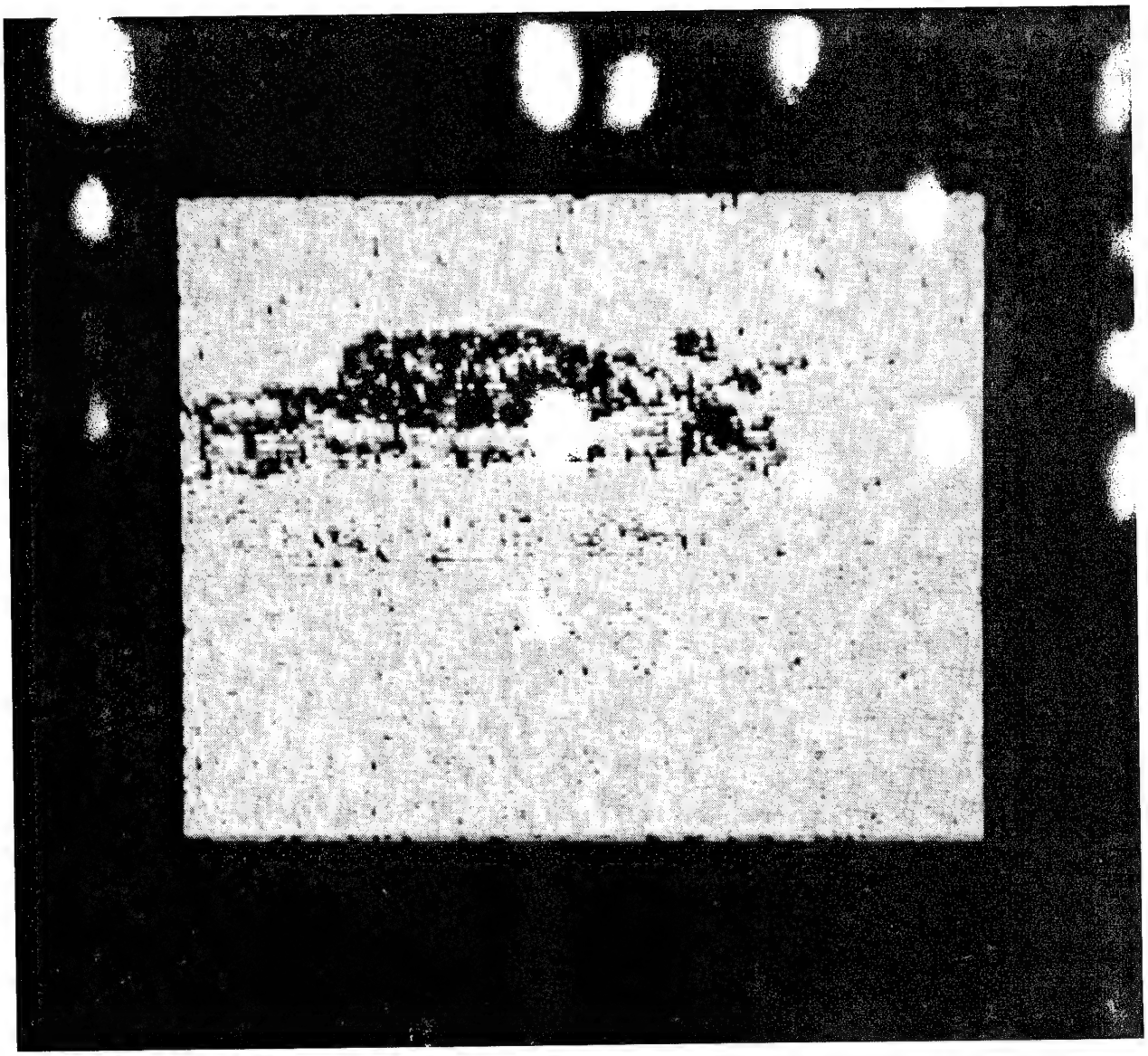


Figure 20. The Shifted and Noisy m60d20.090 (OS) in a Medium Intensity Background

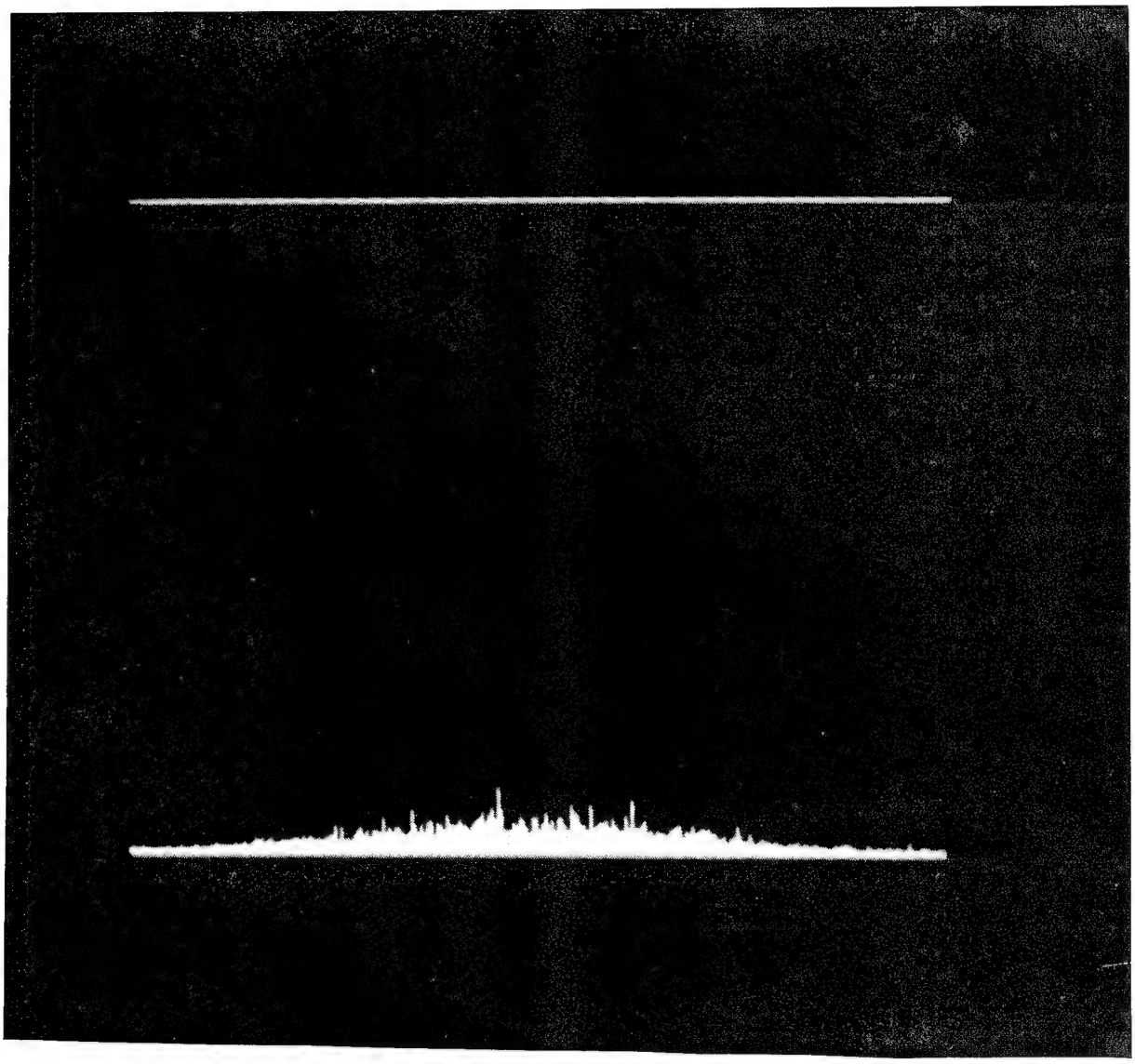


Figure 21. South-to-North Correlation Plane View, the Image of Figure 20 vs. $\mathcal{F}(\mathrm{h})^{*}_{15}$

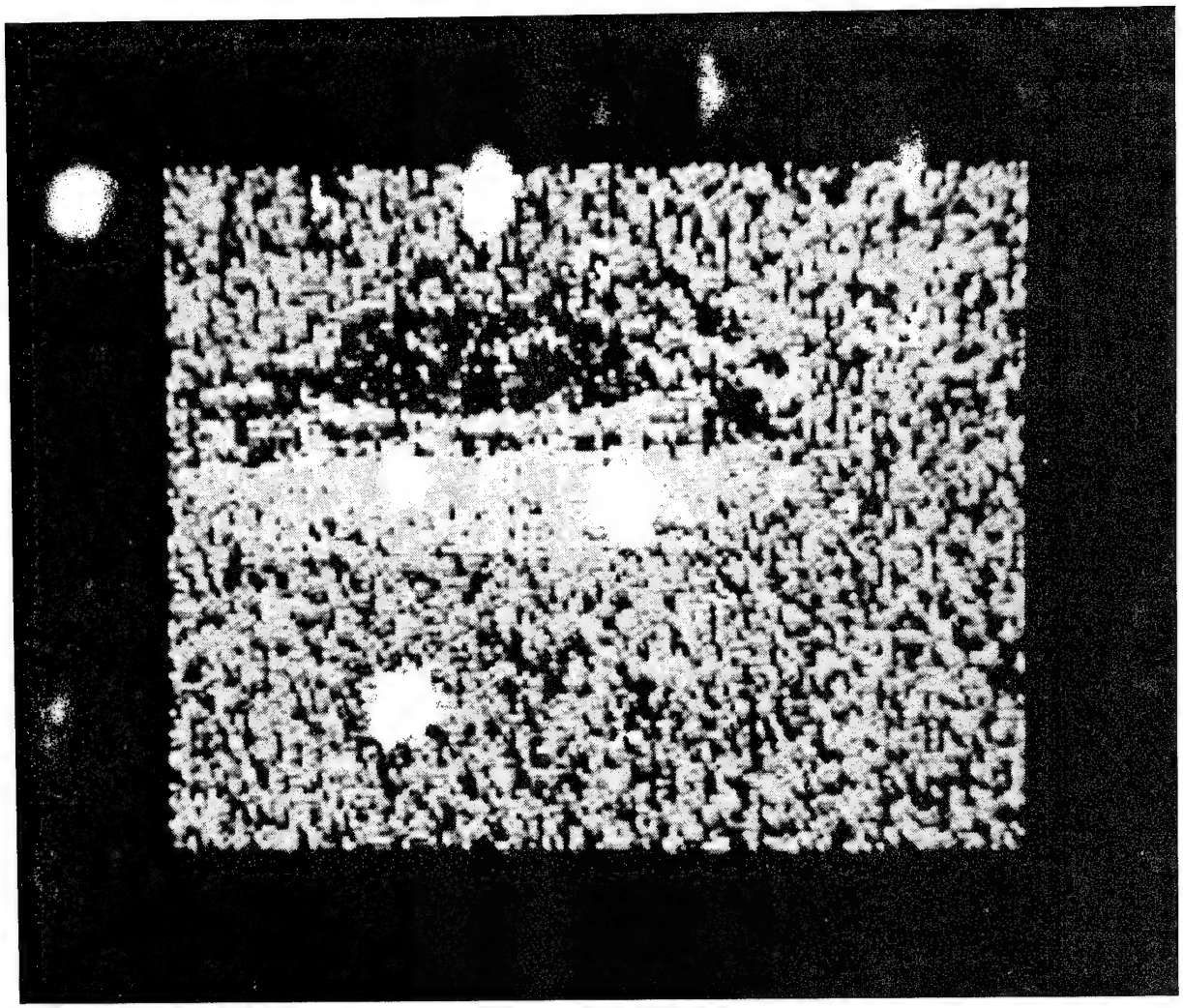


Figure 22. The Shifted and Noisy m60d20.090 (OS) in a Low Intensity Background

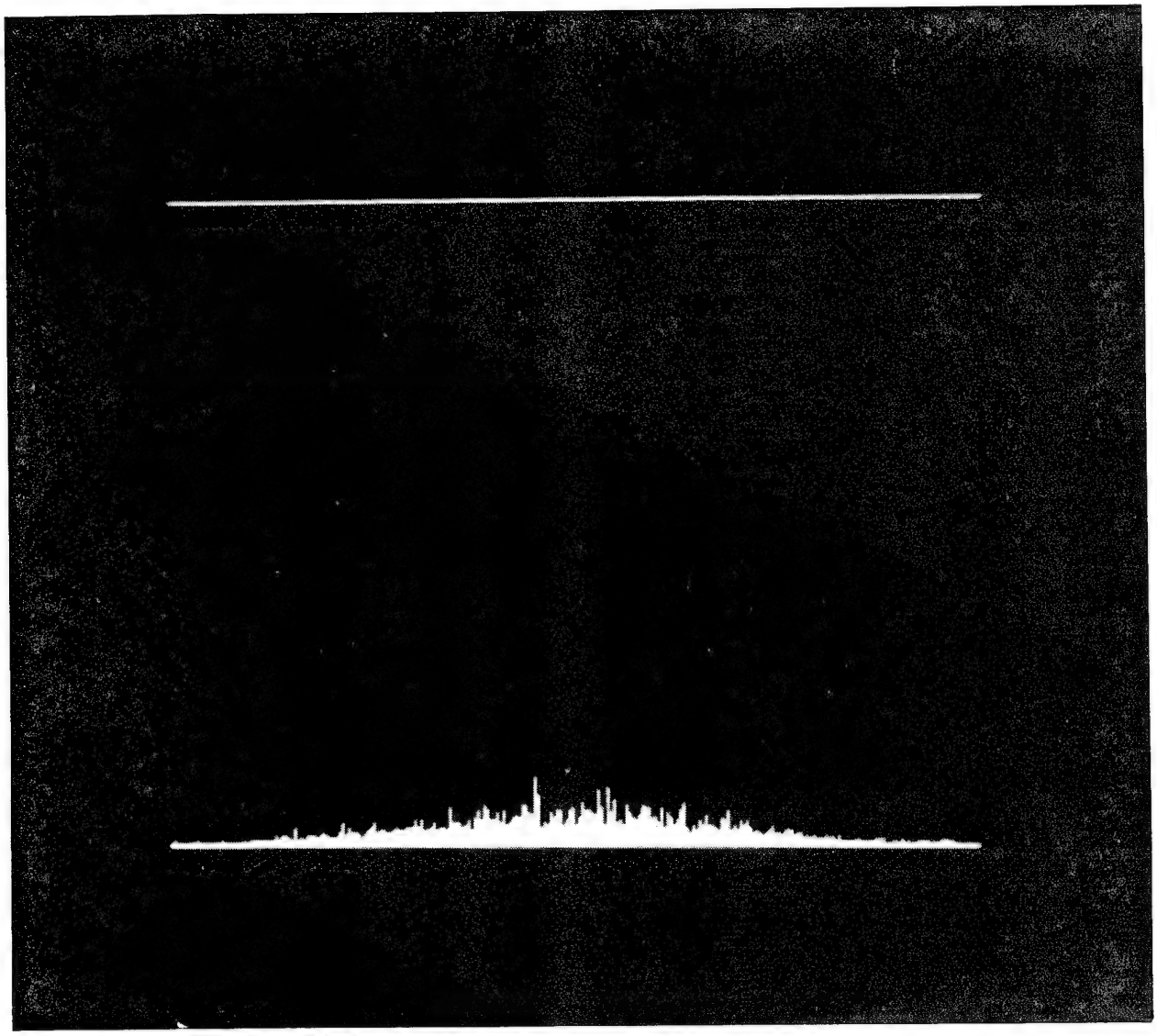


Figure 23. South-to-North Correlation Plane View, the Image of Figure 22 vs. $\mathcal{F}(\mathrm{h})^{*}_{15}$

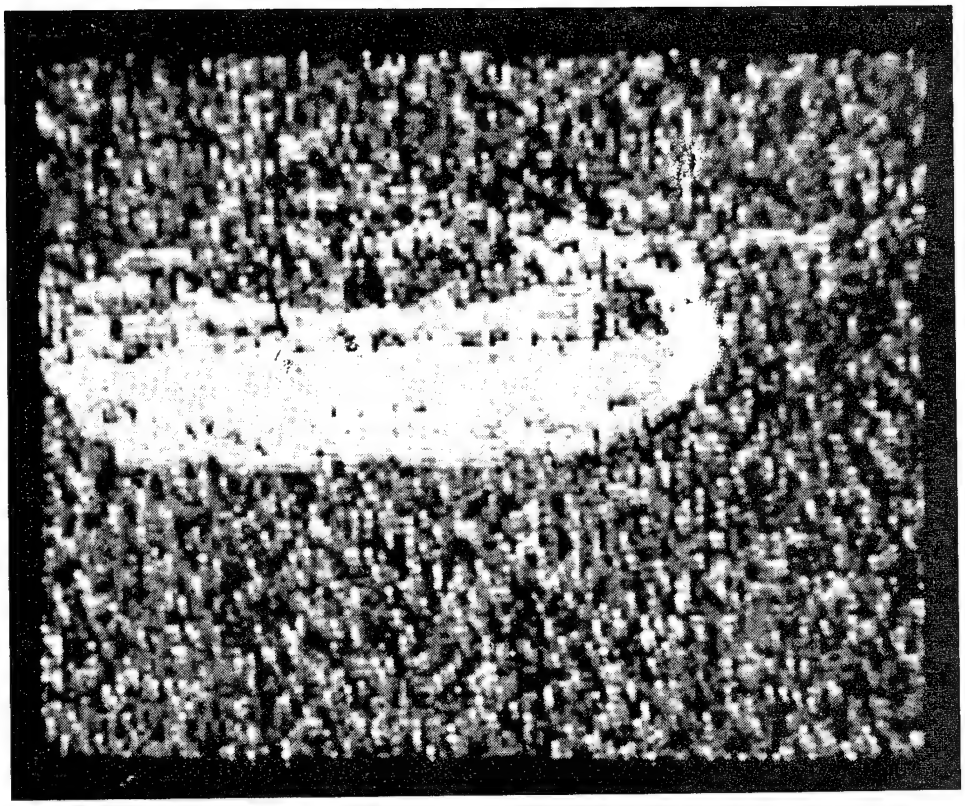


Figure 24. The Shifted and Noisy m60d20.090 (OS) in a High Intensity Background

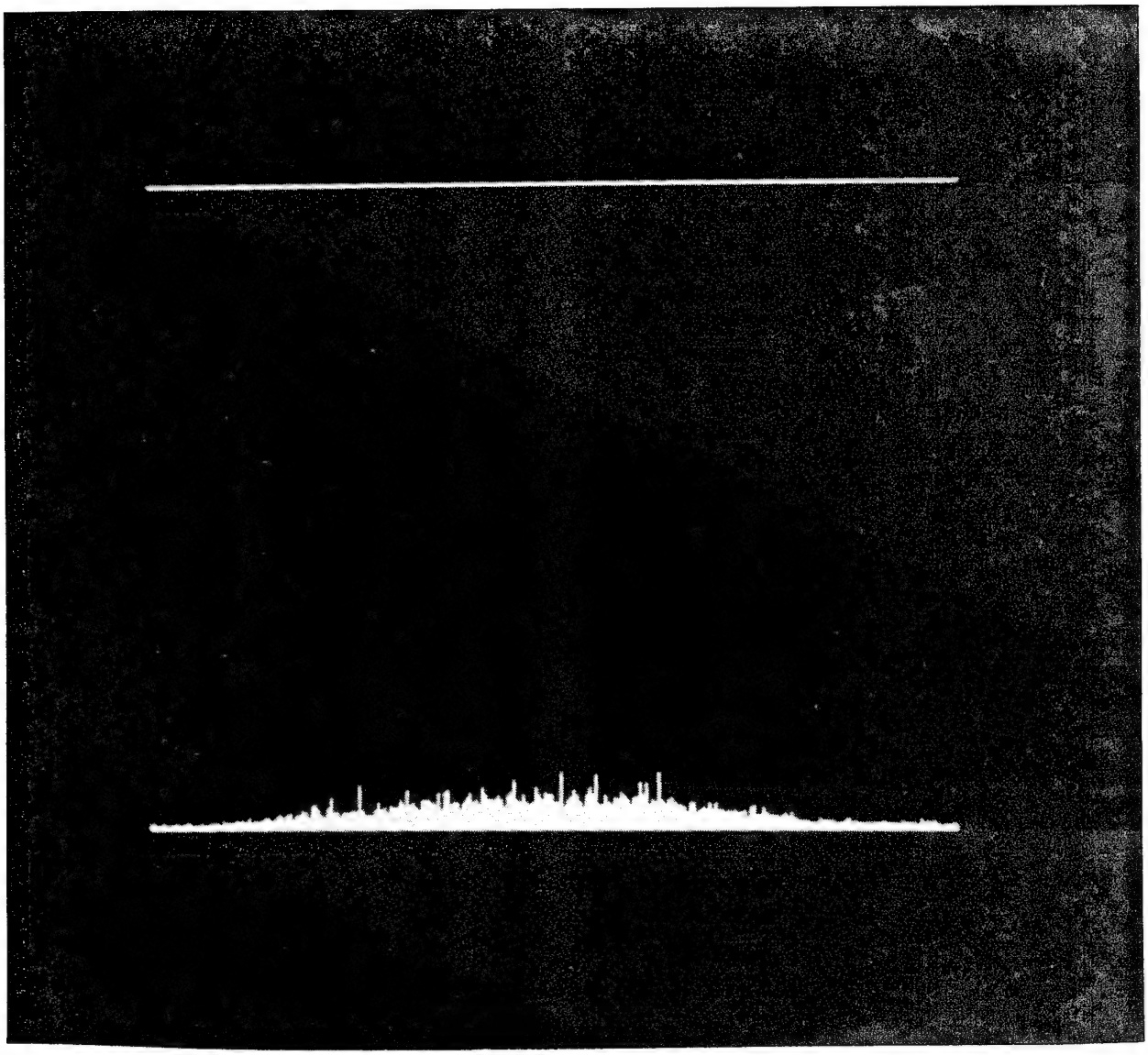


Figure 25. South-to-North Correlation Plane View, the Image of Figure 24 vs. $\mathcal{F}(\mathrm{h})^{*}_{15}$

intensity, to translation of the target with respect to the background, and with respect to large amounts of additive zero mean Gaussian noise. The lack of any recognition signal in the three views of the correlation plane corresponding to M60 targets shows that the strong recognition signals in the three views of the correlation plane corresponding to the M48 targets are not mere fortuitous artifacts.

XVII A NEW EDGE ENHANCEMENT ALGORITHM

The current crop of spatial light modulators are constrained to have only two amplitude states, zero and one, for use in the input plane of an optical correlator. Even if this constraint did not exist, there still would be an academic interest in the potential uses of binary imagery. This interest is heightened because of the target/background contrast problem and the translation invariance problem discussed in Sections XIV and XV. The author was therefore compelled to consider a variety of image binarization techniques during this effort.

Most binarization algorithms involve a mixture of local and/or global thresholding and/or histogram techniques. One quickly realizes that applying these techniques to raw input imagery is full of pitfalls and unsatisfactory results. One instead should apply these algorithms not to the original image, but to a processed version of the original image. One instead is forced to focus one's attention on edge enhancement methods. Two popular edge enhancement algorithms are the diff3 algorithm and the Sobel algorithm. These seem satisfactory for most purposes and can be implemented in real time, though the author has found the performance of the Sobel to be superior in most respects. During the course of this work, however, Elaine Pettit (a Ph.D. candidate in Computer Science at UNT working under the author) and the author discovered a novel edge enhancement scheme with some rather curious properties. At the moment this new technique is rather compute intensive, taking about 1.5 seconds for a 128×128 binary image using the author's transputer network for a reasonable choice of the parameters controlling the calculation. So at the moment this new edge enhancement technique is not suitable for real time applications such as a flight test, for example, but it is suitable for processing training set imagery.

An intensity phase-encoding algorithm for byte imagery was discussed in Section XIII. As was noted in previous sections, the use of phase-encoded imagery seems to lead to superior performance in correlation. This suggested a very general hitherto unsuspected phenomenon to the author, viz., that such a recoding might have applications in conventional digital image processing and that all digital image processing algorithms should be examined to see if they are enhanced by meshing them with intensity phase-encoding. In her dissertation Ms. Pettit has intensively studied local Wigner transforms for a wide variety of digital

image processing applications. Wigner transforms are currently quite fashionable and are discussed in almost any text de l t wavelets. An excellent survey including a very comprehensive bibliography can be found in Reference 30. It was therefore natural to try and mesh intensity phase-encoding with local Wigner transforms. This was successfully and easily done and incorporated into the author's gp1 graphics codes, which are discussed in Appendix H.

Figures 26 — 28 were created by the gp1 graphics codes. Each figure consists of three images — an original image, the corresponding edge enhanced image, and the corresponding binarized image. In each case the original image is m48d25.000 (NS) or rntable at Range C-52 of Eglin AFB, and in each case the binarized image was created from the corresponding edge enhanced image using a simple histogram/thresholding technique. Figure 26 employs the diff3 edge enhancement algorithm, Figure 27 employs a Sobel edge enhancement technique, and Figure 28 utilizes the local Wigner transform intensity phase-encoded technique. It is the author's opinion that, at least for these images, the local Wigner transform intensity phase-encoded technique led to a superior binarized image. This has been confirmed empirically over a wide range of imagery.

Let f be a target in a zero background, e.g., m48d25.000 (NS), and let g be the image f embedded into a nontrivial background, e.g., m48d25.000 (NS) on a turntable. Though not generally appreciated, a moment's thought shows that most arization techniques plied to f will yield a binary image which is not the binarization of f embedded into the binarization of g. This suggests that the best choice of edge enhancement/binarization preprocessing technique for spatial filter training set preparation is not necessarily that edge enhancement/binarization preprocessing technique used on the input imagery to a correlator. The PI's tentative conclusion is somewhat surprising. The training set imagery should be edge enhanced with the local Wigner transform intensity phase-encoded algorithm and then binarized using the global histogram method. The method to be employed on the input imagery to a correlator is the Sobel edge enhancement technique together with the global histogram binarization method.

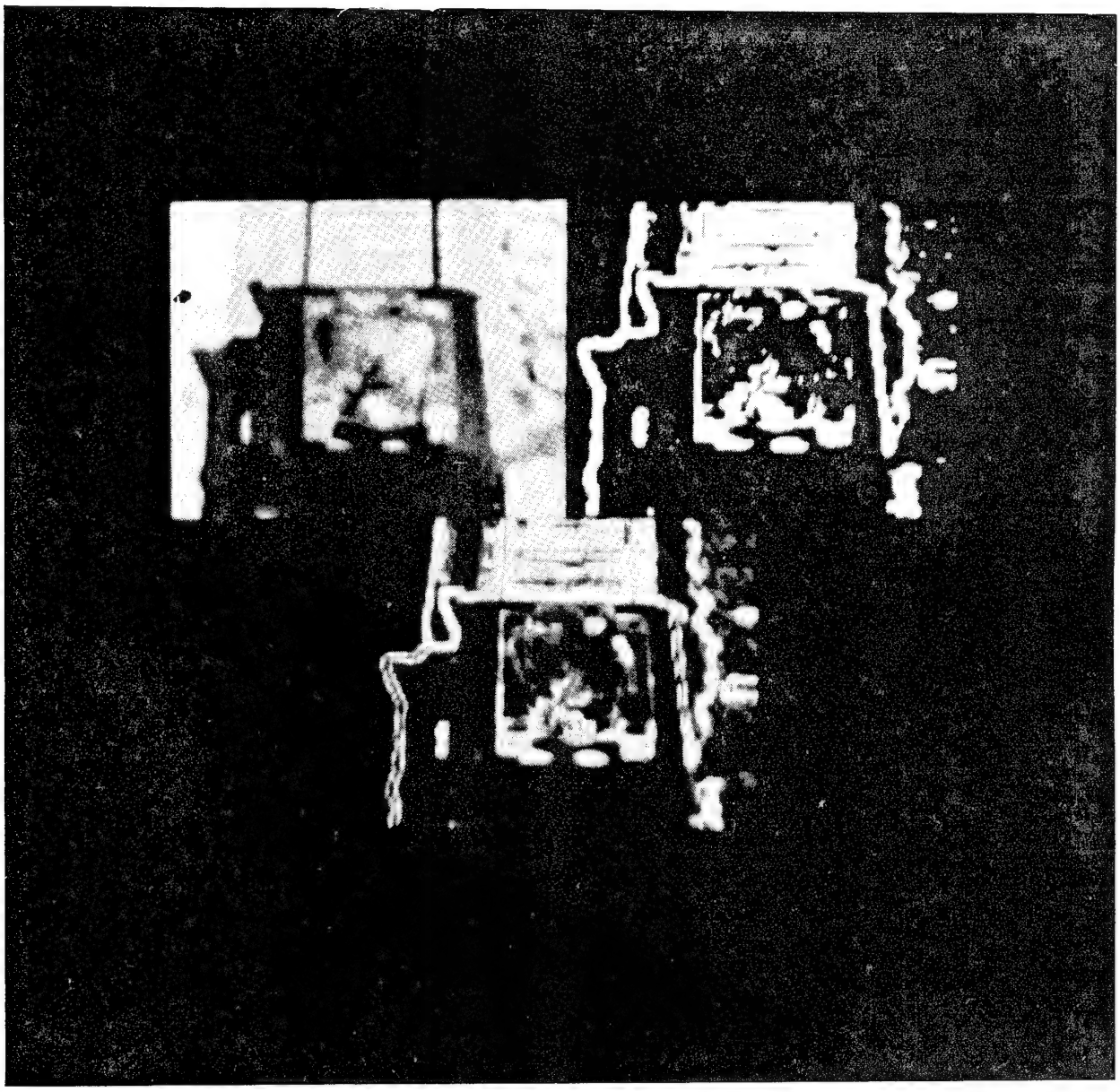


Figure 26. An Example of Diff3 Edge Enhancement

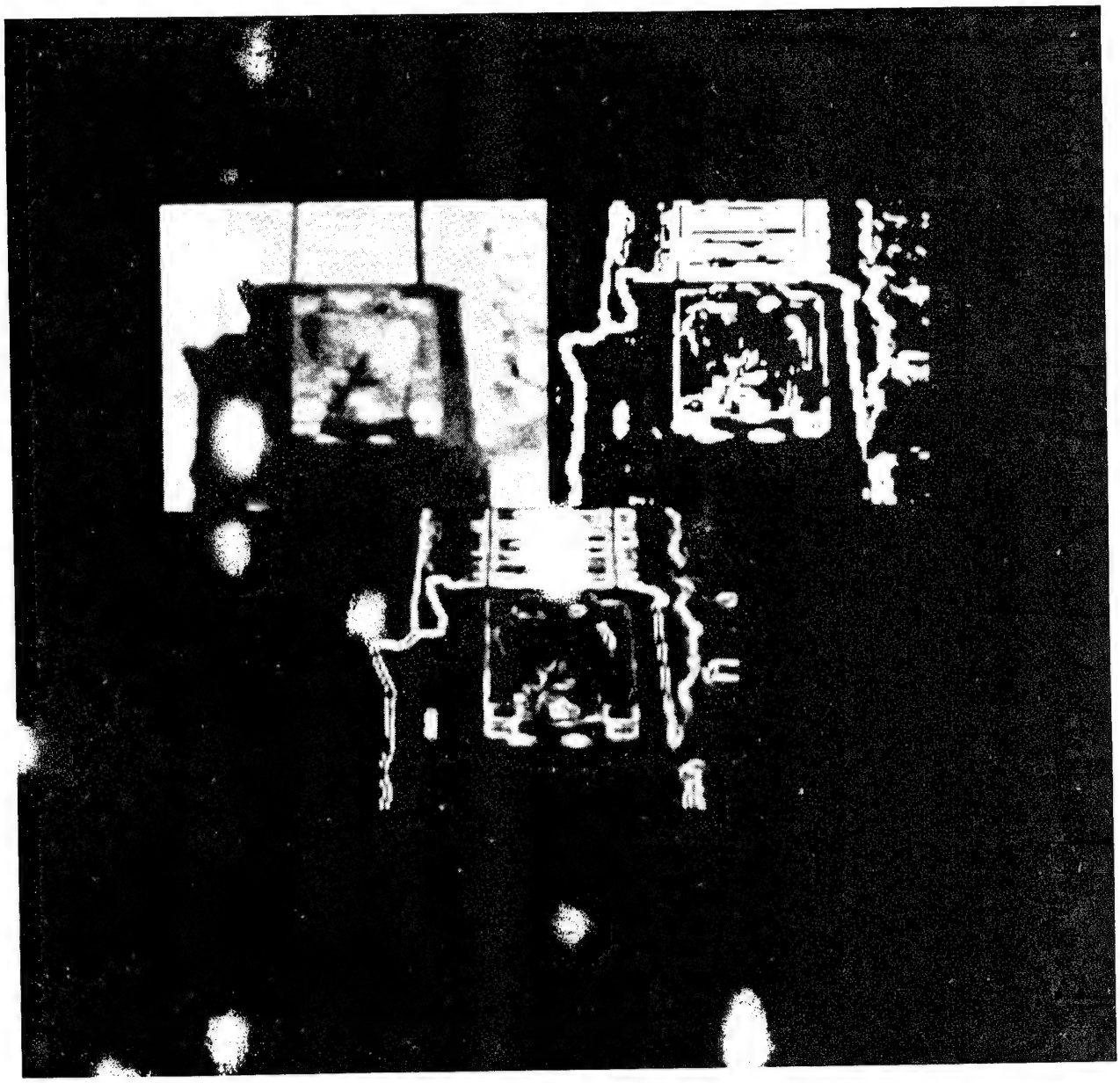


Figure 27. An Example of Sobel Edge Enhancement

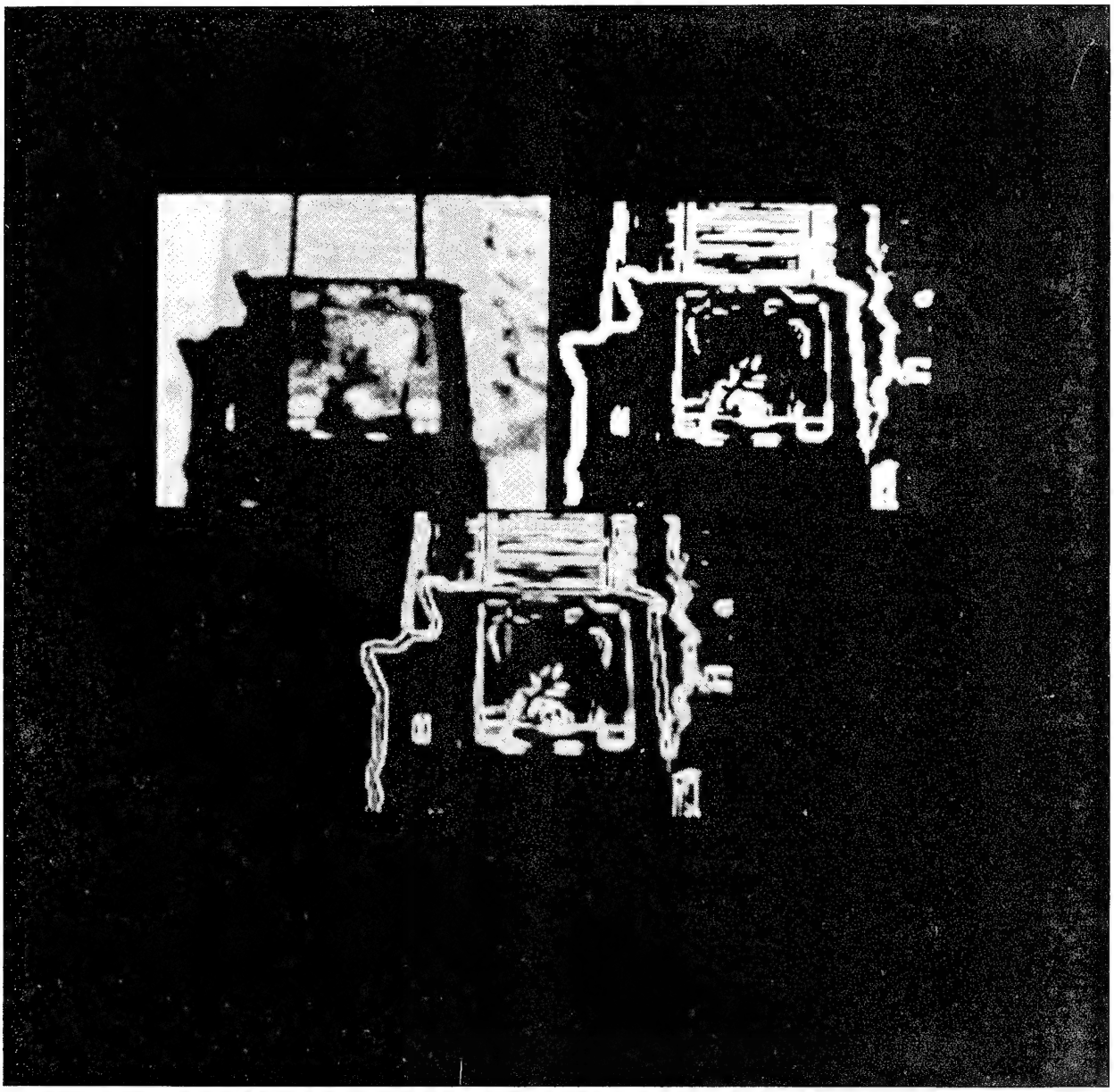


Figure 28. An Example of Local Wigner Transform Intensity Phase-Encoded Edge Enhancement

XVIII CONCLUSIONS

In this effort the author successfully ported, extended, and extrapolated his phase-only filter design codes to the occam 2 parallel programming language and successfully implemented, debugged, and tested them on a network of INMOS transputers.

In this effort the author successfully found algorithms to solve the extremely complicated optimization question involved in his proposed method for designing general spatial filters as a constrained optimization question. These algorithms were successfully implemented in occam 2 codes and debugged, tested, and used on networks of INMOS transputers.

Another goal of this effort was to create an algorithm to determine in a reasonable manner the region of support in a spatial filter whose entries are either zero or a complex number of modulus one. This has been accomplished.

An assigned task of this effort was to deliver to Wright Laboratory Armament Directorate a user friendly version of the author's spatial filter design codes together with a comprehensive User Guide. These items have been delivered to Wright Laboratory. The User Guide to the author's spatial filter design codes are given in Appendix A — Appendix G

Processing of target imagery and testing the performance of spatial filters necessitated the creation of a variety of graphics programs. These were written completely from scratch to drive a SONY GDM-1953 graphics color monitor. These graphics programs have proven invaluable in viewing, manipulating, edge enhancing, and binarizing imagery and in simulating the optical correlation process in great detail. An occam 2 program to drive a transputer based framegrabber system was also created. The characteristics and capabilities of these graphics software tools are sketched in Appendix H.

Circa 1985 the author noted two difficulties in spatial filtering which apparently had not been noted up to that time. The first difficulty arises from variations in the target/background contrast. The second arises when a target in a nontrivial background is translated with respect to the background. These difficulties are discussed in Section XIV and Section XV. A goal of this effort was to devise a reasonably effective solution to these two problems while preserving the advantages of correlation. This has been accomplished. A tentative solution is discussed in Section XVI.

Furthermore, a large part of the labor during this effort consisted of the design and construction of large numbers of spatial filters which were forwarded to Wright Laboratory. During the course of this effort thousands of spatial filters were delivered. The author believes that all requests for spatial filters were answered rapidly and that the resulting spatial filters more than met all design requirements.

XIX RECOMMENDATIONS

The following is a small list of tasks out of the many which should be completed in the future.

An optimization algorithm and corresponding occam 2 codes are needed to determine the threshold used in finding the region of support in a ZAP filter rather than the user prescribed ad hoc choice of threshold described in Section XI. Work on this has already commenced and in fact is near completion.

The author's spatial filter design codes should be translated to ANSI C for use on modern workstations. The author's graphics codes should also be ported to ANSI C code with the graphics per se handled in an X Window format.

The possible use of correlators to find target-like objects in a scene should be investigated. This would likely be much faster than digital methods and should ease the more delicate and compute intensive final target recognition question.

Much more testing needs to be done to see if the advantages of intensity phase-encoding imagery apparent in simulations carries over to hardware.

Good compact discrete ZAP filters can be made which have high degrees of invariance with respect to changes in elevation, azimuth, and pitch in the target training set. Designing very good spatial filters which are additionally invariant with respect to changes in scale puts a bit of stress on the performance of the resulting filters. A serious attempt should be made to find optical/digital techniques to scale images to various levels before they are used as inputs to a correlator.

There is a possibility that new and improved algorithms exist for designing general spatial filters which are more efficient than the algorithms studied in Section X. This possibility should be investigated.

APPENDIX A

AN INTRODUCTION TO THE DOCUMENTATION

Appendices B-G constitute a user guide to the software developed by Robert R. Kallman, Department of Mathematics, University of North Texas, P.O. Box 5116, Denton, Texas 76203-5116 under contract F08635-90-K-0102 with Wright Laboratory Armament Directorate, Eglin Air Force Base, Florida 32542-5434. Dr. Dennis H. Goldstein, WL/MNGA, managed the program for Wright Laboratory. The work was conducted during the period from September 20, 1990 through September 30, 1994.

The subsequent appendices B-G are essentially duplicates of the documents which accompany the software tools. The reader should start with Appendix B for an overview of the software tools and proceed from there.

The author would be happy to hear from any user who has found an error or is confused about nebulous explanations. These comments will be taken into account in later editions of this user manual.

The author wishes to thank Captain Eric P. Augustus, WL/MNGA, for much good practical advice on the software tools developed during this project.

APPENDIX B

OVERVIEW.DOC

1.0. **Introduction and Purpose**. The purpose of this document is to give a very general overview of the software tools created by Robert R. Kallman for spatial filter design.

2.0. **Some Comments on Spatial Filters**. All of the spatial filters designed by the software tools can be used in a digital correlator. Some of the specially designed spatial filters are suitable for use in optical correlators, many of which have strong constraints on the type of spatial filters which they can implement. For example, many spatial light modulators can handle only spatial filters which consist of arrays of +1s or -1s or, in some instances, arrays of +1s, -1s, and 0s. Even in the perfect situation in which a spatial light modulator can implement an arbitrary array of complex numbers, storage requirements for the library of filters may well require constraints on the type of filter used. For example a full complex N x N spatial filter would in general require $8N^2$ bytes of storage, whereas a zero and fifteen phase state filter requires only $N^2/2$ bytes of storage. Optical signal processing systems which can implement filters of the latter sort will soon be available. Moreover, full complex spatial filters of an unsophisticated sort, for example matched filters, are definitely not the panacea generally believed in realistic cases in which objects of interest are embedded in complicated and varying backgrounds. Hence, one is forced to consider other spatial filters. Suitably designed zero and fifteen phase state filters have proved to possess virtually every good characteristic of full complex filters and, for suitably modified inputs, to be approximately invariant with respect to background changes and translations with respect to the background. Therefore, filters whose entries are allowed to be zero and at most fifteen phase states are recommended in almost every circumstance.

3.0. **Hardware Requirements**. The software tools must be used with a parallel processing system consisting of INMOS compatible transputer modules linked together in a linear pipeline. The tools require certain instructions that are available only on transputers with a floating point processor. Hence, the transputer on each module should be a T800, T801, T805, or T9000 transputer. In general more memory is required for the calculations

done on the root module than on the others. The user should check the *.map files generated during compilation to make sure that his specific system has enough memory on each module for the task contemplated. A more detailed discussion of each tool's use is contained in the documentation specific to that tool.

4.0. **Software Requirements**. The software tools discussed here must be used with the INMOS D7305A or compatible occam 2 toolset. This is necessary for trouble free compiling, linking, collecting, configuring, and running of the software tools.

5.0. **The Software Tools**. The most general full blown design of a zero and fifteen phase state filter requires the following tools in the following order:

1. df codes;
2. dc5 codes;
3. dc8 codes;
4. fanal codes;
5. dc5 codes;
6. dc5z codes.

A variety of data files are mentioned in the course of discussing these codes. Many of these data files will be written to or are assumed to lie in a user created \temp\ subdirectory. The names for all of the files mentioned and their paths are user selectable.

5.1. **Step #1, the df codes**. In Step #1 the df codes, assumed to lie in the \df\ subdirectory, are used to create three data files -\temp\target.dat, \temp\cpbmkey.dat, and \temp\tarwt.dat which are used in steps 2, 3, 5, and 6. More importantly the df codes generate a crude initial guess for a good spatial filter (usually and more specifically, a zero and continuous phase filter) which is written to the file \temp\pofa.tmp. The user created input file \temp\fbm.dat controls the choice of which pixels in pofa.tmp are set equal to zero.

5.2. **Step #2, the first use of the dc5 codes**. The dc5 codes, assumed to lie in the \dc5\ subdirectory, should be used in Step #2 with the compile time parameter

false.target.number set equal to zero. Copy \temp\pofa.tmp to \temp\pof.tmp. This is done since it is sometimes useful to retain the initial guess pofa.tmp generated by the df codes in an unaltered file. The dc5 codes use \temp\target.dat, \temp\tarwt.dat, \temp\cpbmkey.dat, and, in this step, \temp\fbm.dat. The dc5 codes start with the initial guess for \temp\pof.tmp and then iteratively generate improved versions of \temp\pof.tmp. The input file \temp\fbm.dat controls the choice of which pixels in \temp\pof.tmp are set equal to zero. These zero pixels are not altered during this optimization. All other (nonzero) pixels are complex numbers of modulus one (i.e., phases). The optimization proceeds by varying these phases in an intelligent manner to drive the threshold for the filter to as high a level as possible. In other words, the smallest recognition signal coming off the true target set is driven to as high a level as possible.

5.3. **Step #3, the dc8 codes**. The dc8 codes are assumed to lie in the \dc8\ subdirectory. Copy \temp\pof.tmp to temp\filter.tmp. The dc8 codes use \temp\target.dat, \temp\tarwt.dat, \temp\cpbmkey.dat, and \temp\fbm.dat. The dc8 program iteratively generates improved versions of \temp\filter.tmp. The input file \temp\fbm.dat controls the choice of which pixels in \temp\filter.tmp are permanently fixed to be zero. All other (nonzero) pixels are complex numbers of nonzero modulus less than or equal to one. The optimization proceeds by varying the phases and the amplitudes, constrained to be between zero and one, in an intelligent manner to drive the SCR for \temp\filter.tmp to as high a level as possible. When dc8 is complete, \temp\filter.tmp usually contains a wide distribution of amplitudes between zero and one in its entries, even though these amplitudes started with a value of either zero or one.

5.4. **Step #4, the fanal program**. The fanal (= filter analysis) program is assumed to lie in the \fanal\ subdirectory. The use of this program only makes sense with the output \temp\filter.tmp of the dc8 codes. The fanal program starts with \temp\filter.tmp and a user defined threshold and outputs three files: \temp\fbm001.tmp, \temp\fil001.tmp, and \temp\pof001.tmp. The file \temp\fbm001.tmp, an array of booleans, has entry which is FALSE if the corresponding entry in \temp\filter.tmp has an amplitude less than or equal to the threshold and has an entry which is TRUE otherwise. Note that if an entry of

\temp\fbm.dat is FALSE then the corresponding entry of \temp\fbm001.tmp is FALSE. The file \temp\fil001.tmp is a spatial filter created from \temp\filter.tmp by setting an entry of \temp\fil001.tmp equal to zero if the corresponding entry of \temp\fbm001.tmp is FALSE and by setting an entry of \temp\fil001.tmp equal to the corresponding entry in \temp\filter.tmp otherwise. The file \temp\fil001.tmp will be of no use to us in this discussion. The file \temp\pof001.tmp, a zero and continuous phase filter, is obtained from \temp\filter.tmp by setting an entry of \temp\pof001.tmp equal to zero if the corresponding entry of \temp\fbm001.tmp is FALSE and by setting an entry of \temp\pof001.tmp equal to the phase of the corresponding entry of \temp\filter.tmp otherwise.

5.5. **Step #5, the second use of the dc5 codes.** Copy \temp\pof001.tmp to \temp\pof.tmp and copy \temp\fbm001.tmp to fbm.dat. In this step the dc5 codes make use of the files \temp\target.dat, \temp\tarwt.dat, \temp\cpbmkey.dat, and \temp\fbm.dat. Also, in this step the dc5 codes make full use of the false target information. The dc5 codes start with \temp\pof.tmp and then iteratively generate improved versions of \temp\pof.tmp. The input file \temp\fbm.dat controls the choice of which pixels in \temp\pof.tmp are set equal to zero. These zero pixels are not altered during this optimization. All other (nonzero) pixels are complex numbers of modulus one (i.e., phases). The optimization proceeds by varying these phases in an intelligent manner to drive the signal-to-clutter ratio (SCR) for the filter to as high a level as possible.

5.6. **Step #6, the dc5z codes.** The dc5z codes are assumed to lie in the \dc5z\ subdirectory. The dc5z codes use \temp\pof.tmp, \temp\target.dat, \temp\tarwt.dat, \temp\cpbmkey.dat, and \temp\fbm.dat. The dc5z codes optimally discretize \temp\pof.tmp into a filter whose entries are zero and a finite number of equally spaced phase states. The number of equally spaced phase states is user selectable. The output of the dc5z codes can be found in the file \temp\pod.tmp.

APPENDIX C

DF.DOC

1.0. **Introduction and Purpose**. The purpose of this document is to give a very general overview of the df (data file) programs created by Robert R. Kallman. These codes create a variety of data files needed as inputs to the dc5, the dc8, and the dc5z spatial filter design codes. The user should first read the file overview.doc for a view of where the df programs fit into the general scheme of filter design. Some of the notation used in overview.doc is also used here.

2.0. **Contents of df.zip**. The df tools are usually supplied to the user in the binary file df.zip. This binary file is created by the data compression utility pkzip, version 2.04g. The following is a list of the files which will be created by applying the pkunzip utility to df.zip:

fbm.bas
cleanup.bat
df.bat
godf.bat
df.dat
df.doc
overview.doc
fbm.fun
comp0.inc
df.inc
dfprot.inc
df.lib
df.lnk
dfroot.occ
dfworker.occ
df.pgm

!!!WARNING!!! The user is advised to leave all of these files unaltered with the exceptions

of comp0.dat and df.inc. Completely unpredictable, untoward, and erroneous results will be a consequence of not heeding this caveat.

3.0. **Overview of the df Programs**. The df programs are assumed to lie in the \df\ subdirectory. Unless stated otherwise all of the files listed in subsection 2.0 must lie in this subdirectory. The df codes are used to create three data files -\temp\target.dat, \temp\cpbmkey.dat, and \temp\tarwt.dat. These data files are needed as inputs to the dc5, the dc8, and the dc5z spatial filter design codes. More importantly the df codes generate a crude initial guess for a good spatial filter (usually and more specifically, a zero and continuous phase filter) which is written to the file \temp\pofa.tmp. The user created input file \temp\fbm.dat controls the choice of which pixels in pofa.tmp are set equal to zero.

4.0. **Hardware Requirements**. The df programs must be used with a parallel computing system consisting of INMOS compatible transputer modules linked together in a linear pipeline. Transputers with a floating point processor are needed since the df codes use certain instruction which are available on such transputers. Hence, the transputers should be a T800, T801, T805, or T9000 transputers. It is assumed that there are 8 megabytes of memory on the transputers. If not, minor changes must be made to the otherwise untouchable file df.pgm. For example, suppose the user has 4 megabytes of memory on the transputers of his system. Using any text editor change the line

SET processor (type, memsize :="T800", (8 TIMES M))

to

SET processor (type, memsize :="T800", (4 TIMES M)).

If the user's system has a mix of transputer modules with varying amounts of memory, then altering df.pgm is only slightly more complicated. The user should read the appropriate sections of the D7305A occam 2 toolset documentation for a detailed discussion of the syntax used in the hardware description contained in df.pgm.

The user should check the df.map file generated during compilation to make sure that there is sufficient memory on each of his modules to accommodate his computational requirements.

5.0. **Software Requirements**. This software tool must be used with the INMOS D7305A or compatible occam 2 toolset. This is necessary for trouble free compiling, linking, collecting, configuring, and running of the software tool.

6.0. **Editing comp0.inc**. The df programs use three include files, comp0.inc, df.inc, and dfprot.inc, all of which are listed in subsection 2.0. Two of these include files, df.inc and dfprot.inc, must not be altered under any circumstances. The include file comp0.inc contains constants which must be known before compilation. For example, the language occam 2 needs to know all array sizes during compilation. This file must be in the \df\ subdirectory and its name must not be altered.

6.1. **VAL INT number.of.processors**. The 32 bit integer number.of.processors must be a positive integer less than or equal to the number of transputer modules in the user's linear pipeline.

6.2. **VAL INT input.m**. The 32 bit integer input.m is the number of rows in the byte input images. The current version of the spatial filter design codes require that input.m be a power of 2 and that input.m be equal to input.n.

6.3. **VAL INT input.n**. The 32 bit integer input.n is the number of columns in the byte input images. The current version of the spatial filter design codes require that input.n be a power of 2 and that input.n be equal to input.m.

6.4. **VAL INT ft.m**. The 32 bit integer ft.m is the number of rows in the complex array on which two-dimensional full complex fast Fourier transforms are performed. The current version of the df programs require that ft.m be a power of 2 and that ft.m be the same as ft.n. Also, the current version of the spatial filter design codes restrict the value of ft.m to be either input.m or 2×input.m.

6.5. **VAL INT ft.n**. The 32 bit integer ft.n is the number of columns in the complex array on which two-dimensional full complex fast Fourier transforms are performed. The current version of the df programs require that ft.n be a power of 2 and that ft.n be the same

as ft.m. Also, the current version of the spatial filter design codes restrict the value of ft.n to be either input.n or 2×input.n.

6.6. **VAL INT filter.m**. The 32 bit integer filter.m is the number of rows in the complex array representing the spatial filter. The current version of the spatial filter design codes require that filter.m be a power of 2, that filter.m be equal to filter.n, and that filter.m be equal to input.m.

6.7. **VAL INT filter.n**. The 32 bit integer filter.n is the number of columns in the complex array representing the spatial filter. The current version of the spatial filter design codes require that filter.n be a power of 2, that filter.n be equal to filter.m, and that filter.n be equal to input.n.

6.8. **VAL INT maximum.number.of.image.labels**. This 32 bit integer must be greater than or equal to 1. One of the modes to enter imagery to the df programs permits one to choose selected entries from files of concatenated byte images (no headers allowed). These labels will be read into an array of integers. The integer maximum.number.of.image.labels then has a self evident meaning. It is necessary to specify this integer at compilation time since the D7305A toolset needs to know array sizes at compilation time. This number should not be overly large otherwise memory is wasted.

6.9. **VAL INT maximum.number.of.input.filenames**. This 32 bit integer must be greater than or equal to 1. The byte images used as inputs to the df codes can be spread over an arbitrarily large number of files. The integer maximum.number.of.input.filenames then has a self evident meaning. It is necessary to specify this integer at compilation time since the D7305A toolset needs to know array sizes at compilation time. This number should not be overly large otherwise memory is wasted.

6.10. **VAL []BYTE rddfdat.filename**. The linear byte array rddfdat.filename is the name which is internal to the program for what we here call df.dat (discussed in 8.0). The df program needs to know where to find df.dat at compilation time but not its contents. If the file df.dat is located in the \df\ subdirectory, then no pathname is needed. If the df.dat

file is located elsewhere, the complete pathname must be included in rddfdat.filename.

7.0. **Creating the Bootable File df.btl**. One creates the bootable file df.btl by invoking the batch file df.bat, i.e., by typing "df". In order for this command to succeed, the include files comp0.inc, df.inc, and dfprot.inc, the library file df.lib, the linker indirect file df.lnk, the occam 2 text files dfroot.occ and dfworker.occ, and the text file df.pgm must be in the \df\ subdirectory. A number of ancillary files are created by invoking df.bat. Do not be alarmed or confused. Only the final product, df.btl, is actually used.

8.0. **Editing df.dat**. The data file df.dat contains constants which are used in the df program but which need not be specified at compile time. However, the location of the df.dat file itself must be known at compile time and is specified by rddfdat.filename located in comp0.inc. The format of df.dat must be preserved. The user should make a backup copy before altering it. There is a large number of parameters which can be altered. Most of the time only a few need be changed. These large number of parameters give the user a great deal of flexibility in choice of correlator input methods and types of filters used in the initial guess.

8.1. **[][80]BYTE input.filename**. List here the names (including complete paths) of the data files which contain the target images. There must be at least one data file listed here. Each data file must be on its own line. There must be no blank lines. The number of characters in the name of each data file cannot exceed 80. The lengths of the names of the data files need not be the same. The upper bound on the number of data files is max.number.of.input.filenames, which is set in comp0.inc. This upper bound can be changed in comp0.inc, but then the program must be recompiled. Other than memory constraints, there is no further worry about the number of filenames involved. The df codes will automatically determine the number of target data files provided the correct format of df.dat is preserved.

8.2. **[]BOOL true.or.false target**. List here booleans (TRUE or FALSE) corresponding to each target data file listed in 8.1, TRUE being listed if the corresponding target data file

consists of true targets and FALSE being listed if the corresponding target data file consists of false targets. There must be only one boolean per line, no lines can be skipped, and the number of booleans must be exactly the same as the number number.of.input.filenames discussed in 8.1.

8.3. []**BOOL sequential.input.mode**. There are two modes in which the data is read from an individual target data file. The sequential input mode is one of these. In this mode a target data file is opened for reading, a certain target is selected, read in, and processed, and then a certain number (perhaps 0) of other targets are read in sequentially, proceeding from one image to the next with a uniform stride of images. List here booleans (TRUE or FALSE) corresponding to each target data file listed in 8.1, TRUE being listed if the corresponding target data file is read in this sequential input mode and FALSE being listed otherwise. There must be only one boolean per line, no lines can be skipped, and the number of booleans must be exactly the same as the number number.of.input.filenames discussed in 8.1.

8.4. []**INT label.of.first.image.in.file**.

List here 32 bit integers corresponding to each target data file listed in 8.1, the integer being the label of the very first image in the file. There must be only one integer per line, no lines can be skipped, and the number of integers must be exactly the same as the number number.of.input.filenames discussed in 8.1.

8.5. []**INT label.of.first.image.used.in.file**.

List here 32 bit integers corresponding to each target data file listed in 8.1, the integer being the label of the first image read in the file if the file is to be read in the sequential input mode, 0 (or any other 32 bit integer) otherwise. There must be only one integer per line, no lines can be skipped, and the number of integers must be exactly the same as the number number.of.input.filenames discussed in 8.1.

8.6. []**INT image.number**. List here 32 bit integers corresponding to each target data file listed in 8.1, the integer being the number of images read in the file if the file is to be

read in the sequential input mode, 0 (or any other 32 bit integer) otherwise. There must be only one integer per line, no lines can be skipped, and the number of integers must be exactly the same as the number number.of.input.filenames discussed in 8.1.

8.7. **[]INT image.stride**. List here 32 bit integers corresponding to each target data file listed in 8.1, the integer being the stride through the file if the file is to be read in the sequential input mode, 0 (or any other 32 bit integer) otherwise. If every target is to be read in after the initial target is read, then the corresponding line here should be 1. If every third target is to be read in after the initial target is read in, the corresponding line here should be 3. Etc. There must be only one integer per line, no lines can be skipped, and the number of integers must be exactly the same as the number number.of.input.filenames discussed in 8.1.

8.8. **[]BOOL individual.label.input.mode**. There are two modes in which the data is read from an individual target data file. This is the second mode (cf. the sequential.input.mode in 8.3), so individual.label.input.mode :=NOT sequential.input.mode. In this mode a target data file is opened for reading, a file of image labels is read (cf. image.label.filename in 8.9), and then the images corresponding to the labels are successively read in and processed. List here booleans (TRUE or FALSE) corresponding to each target data file listed in 8.1, FALSE being listed if the corresponding target data file is read in the individual label input mode.

There must be only one boolean per line, no lines can be skipped, and the number of booleans must be exactly the same as the number number.of.input.filenames discussed in 8.1.

8.9. **[][80]BYTE image.label.filename**. List here the names (including complete paths) of the image label filenames corresponding to each target data file. Each file must be on its own line. There must be no blank lines. The number of characters in the name of each data file cannot exceed 80. The lengths of the names of the data files need not be the same. There must be a nontrivial name if the corresponding target data file is to be read in the individual label input mode. If the corresponding target data file is to be read in the

sequential input mode, write " " for the image label filename. There must be only file name per line, no lines can be skipped, and the number of file names must be exactly the same as the number number.of.input.filenames discussed in 8.1.

8.10. **[]INT background.intensity**. List here 32 bit integers corresponding to each target data file listed in 8.1, the integer representing the intensity of the background in which the target is to be placed. Since the df codes deal with byte images, each byte representing an intensity, the integers here should be between 0 and 255. There must be only one integer per line, no lines can be skipped, and the number of integers must be exactly the same as the number number.of.input.filenames discussed in 8.1.

8.11. **INT false.target.number**. This 32 bit integer is the total number of false targets under consideration.

8.12. **INT true.target.number**. This 32 bit integer is the total number of true targets under consideration.

8.13. **INT target.number**. This 32 bit integer is the total number of targets under consideration. Except in the rarest of experimental situations target.number =false.target.number +true.target.number.

8.14. **[80]BYTE target.output.filename**. This binary file is needed by the dc5, dc8, and dc5z spatial filter design codes and usually should be named c:\temp\target.dat, for example. The df codes process the raw input imagery in a variety of ways viewed as a two-dimensional complex array, as would be done before the imagery is inserted into a correlator, and then the two-dimensional complex FFT is taken. The results are stored in a contiguous manner in the file c:\temp\target.dat. Precomputing these two-dimensional FFT's saves the design codes an enormous amount of time, far more than is wasted in reading c:\temp\target.dat into the transputer network. The number of characters in target.output.filename cannot exceed 80.

8.15. **BOOL amplitude.input.mode**. The df codes are designed to handle byte im-

agery, where the bytes represent intensity levels in some scaling. This boolean is set to be TRUE if the optical system for which the spatial filters are to be designed takes the Fourier transform of square roots of these initial intensities. A glance at Goodman's book on Fourier optics shows that this is a a very common occurrence. This boolean is set to be FALSE if the optical system for which the spatial filters are to be designed takes the Fourier transform of the original intensity image. This apparently occurs in the Image Synthesis System optical-digital correlator being designed by Essex Corporation.

8.16. **BOOL intensity.input.mode**. This boolean is equal to NOT amplitude.input.mode. The boolean amplitude.input.mode is discussed in 8.15.

8.17. **BOOL standard.input.mode**. This boolean should be set to TRUE if the input imagery is not phase-encoded. It should be set to be FALSE if the input imagery is phase-encoded.

8.18. **BOOL phase.encoded.input.mode**. This boolean is equal to NOT standard.input.mode. The boolean standard input mode is discussed in 8.17.

8.19. **BOOL bool.zero.mean.target.array**. This boolean is set to be TRUE if the complex average of the complex target array is to be subtracted from every entry of the complex target array. It should be set FALSE otherwise.

8.20. **BOOL bool.shift.mode**. This boolean was used for experimental purposes. It is recommended that it always be FALSE.

8.21. **INT filter.scaling**. This 32 bit integer, which must be specified, can assume only the values 1 or 2. If ft.m=filter.m, then filter.scaling must be 1. If ft.m>filter.m, then filter scaling can be either 1 (a half-nyquist filter) or 2 (the full nyquist filter). If filter.scaling is 2, then the spatial filter generated by the df codes will be blown up by a factor of 2, so that each cell of the filter.m×filter.n spatial filter will correspond to a 2×2 cell in the ft.m×ft.n array.

8.22. **[80]BYTE filter.output.filename**. This is the name of the file to which the initial guess for a good spatial filter generated by the df codes is written, for example, c:\temp\fil.tmp. We will use this particular name in discussing later options. The number of characters in filter.output.filename cannot exceed 80.

8.23. **BOOL bool.normalize.filter**. If this boolean is TRUE, then each entry of fil.tmp is multiplied by a positive scalar so that the maximum amplitude of the resulting fil.tmp is 1.0. No operations are performed if this boolean is FALSE.

8.24. **BOOL bool.phase.average.mode**. This boolean controls a bifurcation in the choice of how fil.tmp is created. This choice seems to have very little effect on the performance of the final product created by the dc5, dc8, and dc5z spatial filter design codes. Set this boolean to be TRUE as a default.

8.25. **BOOL bool.pof.filter**. This boolean should be TRUE if the initial guess generated by the df codes is to be a zero and continuous phase spatial filter. It should be FALSE otherwise. It is recommended that it be TRUE in almost every case.

8.26. **BOOL bool.scale.filter**. This boolean is used almost exclusively for experimental purposes. It allows the user to multiply each entry of fil.tmp by the scaling.filter.fraction, which is discussed next. It should almost always be set to be FALSE.

8.27. **REAL32 scale.filter.fraction**. Cf. the discussion of bool.scale.filter. The parameter must always be set to be an IEEE 32 bit floating point number. It should usually be set to be 1.0. It has no effect if bool.scale.filter is FALSE.

8.28. **BOOL bool.symmetrize.filter**. This boolean should be set to be TRUE only if bool.shift.mode is TRUE. It is therefore recommended that this boolean always be FALSE.

8.29. **[80]BYTE filter.bool.mask.filename**. This is a user defined binary file needed by the dc5 (first pass) and dc8 spatial filter design codes and on occasion by the dc5 design codes (second pass). It usually should be named c:\temp\fbm.dat, for example. The file

fbm.dat is a filter.m×filter.n sized array of b[illegible]eans (TRUE or FALSE) whose entries correspond in a one-to-one manner to each entry of the full complex spatial filter generated by the df codes. For convenience call this spatial filter fil.tmp. If the (i,j)-th entry of fbm.dat is TRUE, then the fbm.dat file has no influence over the corresponding (i,j)-th entry in fil.tmp. If the (i,j)-th entry of fbm.dat is FALSE, then the corresponding (i,j)-th entry in fil.tmp is forced to be 0.0. It is very important to note that both fil.tmp and fbm.dat are arranged for two-dimensional FFT convenience. This means that fil.tmp is cyclicly biased in such a manner that the lowest spatial frequency contained in fil.tmp is located in the (0,0)-th entry. Two default fbm.dat files for 128×128 spatial filters are included for the user's convenience. These are the files fbm.fun and fbm.bas. Every entry of fbm.fun is TRUE. Every entry of fbm.bas is TRUE with the exception of the (0,0)-th entry which is FALSE. The number of characters in filter.bool.mask.filename cannot exceed 80.

8.30. **[80]BYTE cpbmkey.output.filename**. This text file is needed by the dc5, dc8, and dc5z spatial filter design codes and usually should be named c:\temp\cpbmkey.dat, for example. The df program generates a default version of this file. The user currently should always use this default file. In the future more sophisticated user defined versions of this file might be of use. The number of characters in cpbmkey.filename cannot exceed 80.

8.31. **[80]BYTE tarwt.output.filename**. This text file is needed by the dc5, dc8, and dc5z spatial filter design codes and usually should be named c:\temp\tarwt.dat, for example. The df program generates a default version of this file. The user currently should always use this default file. In the future more sophisticated user defined versions of this file might be of use. The number of characters in tarwt.filename cannot exceed 80.

9.0. **Running the Program**. If all is suitably prepared, run the df.btl program by invoking the simple batch file godf.bat, i.e., type "godf".

10.0. **Terminating the Program**. The program will terminate on its own, either successfully or in an erroneous state, in which case an error message will probably appear on the PC screen. The most common cause of an error message is the nonexistence of a file the

program was seeking to use. The user can terminate the program by typing "control/break" and then typing "x".

11.0. **Cleaning up the \df\ subdirectory**. The \df\ subdirectory may be cleansed of the new files created by invoking the batch file cleanup.bat, i.e., by typing "cleanup". This is useful for housekeeping purposes. Be aware that df.btl is deleted in this process.

APPENDIX D

DC5.DOC

1.0. **Introduction and Purpose.** The purpose of this document is to give a very general overview of the dc5 (design code number 5) programs created by Robert R. Kallman. The dc5 codes start with the initial guess for a zero and continuous phase-only filter, say \temp\pof.tmp, and iteratively generate improved versions of \temp\pof.tmp. The user should first read the file overview.doc for a view of where the dc5 programs fit into the general scheme of spatial filter design. The user should also be familiar with the contents of df.doc. Some of the notation used in overview.doc and df.doc is also used here.

2.0. **Contents of dc5.zip.** The dc5 tools are usually supplied to the user in the binary file dc5.zip. This binary file is created by the data compression utility pkzip, version 2.04g. The following is a list of the files which will be created by applying the pkunzip utility to dc5.zip:

cleanup.bat

dc5.bat

godc5.bat

cpbm1.dat

dc5.dat

misc0.dat

misc1.dat

misc2.dat

misc3.dat

misc4.dat

misc5.dat

misc6.dat

misc7.dat

miscend.dat

dc5.doc

overview.doc

comp0.inc

dc5.inc

dc5prot.inc

dc5.lib

filename.lis

dc5.lnk

boss5.occ

worker5.occ

dc5.pgm

!!!WARNING!!! The user is advised to leave all of these files unaltered with the exceptions of comp0.dat and dc5.dat. Completely unpredictable, untoward, and erroneous results will be a consequence of not heeding this caveat.

3.0. **Overview of the dc5 Programs.** The dc5 programs are assumed to lie in the \dc5\ subdirectory. Unless stated otherwise all of the files listed in subsection 2.0 must lie in this subdirectory. The dc5 codes make use of the files \temp\target.dat, \temp\tarwt.dat, and \temp\cpbmkey.dat generated by the df codes. The dc5 codes start with an initial guess for a zero and continuous phase filter, say \temp\pof.tmp, and then iteratively generate improved versions of \temp\pof.tmp. The \temp\pof.tmp is created either by the df codes (for the first use of the dc5 codes in a full blown design process) or by the fanal program (the second use of the dc5 codes in a full blown design process). The input file \temp\fbm.datcontrols the choice of which pixels in \temp\pof.tmp are set equal to zero. The \temp\fbm.dat file is created either by the user (for the first use of the dc5 codes in a full blown design process) or by the fanal program (in the second use of the dc5 codes in a full blown design process). The \temp\fbm.dat file is discussed in more detail below. The zero pixels are not altered during the optimization. All other (nonzero) pixels are complex numbers of modulus one (i.e., phases). The optimization proceeds by varying these phases in an intelligent manner to drive the signal-to-clutter ratio (SCR) for the filter to as high a level as possible. Other ancillary files needed by the dc5 programs are discussed below.

4.0. **Hardware Requirements.** The dc5 programs must be used with a parallel computing system consisting of INMOS compatible transputer modules linked together in a linear pipeline. Transputers with a floating point processor are needed since the dc5 codes use certain instruction which are available on such transputers. Hence, the transputers should be a T800, T801, T805, or T9000 transputers. It is assumed that there are 8 megabytes of memory on the transputers. If not, minor changes must be made to the otherwise untouchable file dc5.pgm. For example, suppose the user has 4 megabytes of memory on the transputers of his system. Using any text editor change the line

SET processor (type, memsize :="T800", (8 TIMES M))

to

SET processor (type, memsize :="T800", (4 TIMES M)).

If the user's system has a mix of transputer modules with varying amounts of memory, then altering dc5.pgm is only slightly more complicated. The user should read the appropriate sections of the D7305A occam 2 toolset documentation for a detailed discussion of the syntax used in the hardware description contained in dc5.pgm.

The user should check the dc5.map file generated during compilation to make sure that there is sufficient memory on each of his modules to accommodate his computational requirements.

5.0. **Software Requirements.** This software tool must be used with the INMOS D7305A or compatible occam 2 toolset. This is necessary for trouble free compiling, linking, collecting, configuring, and running of the software tool.

6.0. **Editing comp0.inc.** The dc5 programs use three include files, comp0.inc, dc5.inc, and dc5prot.inc, all of which are listed in subsection 2.0. Two of these include files, dc5.inc and dc5prot.inc, must not be altered under any circumstances. The include file comp0.inc contains constants which must be known before compilation. For example, the language occam 2 needs to know all array sizes during compilation. This file must be in the \dc5\ subdirectory and its name must not be altered.

6.1. **VAL INT bigm.** The 32 bit integer bigm is the number of rows in the complex

array on which two-dimensional full complex fast Fourier transforms are performed. The current version of the dc5 programs require that bigm be a power of 2 and that bigm be the same as bign. Also, the current version of the spatial filter design codes restrict the value of bigm to be either m or 2m. The integer bigm must coincide with the integer ft.m used in the df codes. The notation may be unified in some future version of these tools.

6.2. **VAL INT bign**. The 32 bit integer bign is the number of rows in the complex array on which two-dimensional full complex fast Fourier transforms are performed. The current version of the dc5 programs require that bign be a power of 2 and that bign be the same as bigm. Also, the current version of the spatial filter design codes restrict the value of bign to be either n or 2n. The integer bign must coincide with the integer ft.n used in the df codes. The notation may be unified in some future version of these tools.

6.3. **VAL INT cpbm.number**. This constant has been largely used for experimental purposes. It should always be set to be 1.

6.4. **VAL INT false.target.number**. This 32 bit integer is the total number of false targets in the training set. As noted in overview.doc it should be set to 0 during the first use of the dc5 codes in a full blown spatial filter design.

6.5. **VAL INT m**. The 32 bit integer m is the number of rows in the complex array representing the spatial filter. The current version of the spatial filter design codes require that m be a power of 2 and that m be equal to n. The integer m must coincide with the integer filter.m used in the df codes. The notation may be unified in some future version of the tools.

6.6. **VAL INT max.number.of.program.passes**.

This 32 bit integer should be kept at 9. It provides an upper bound on the integer appearing on the second line of filename.lis, discussed in 9.0.

6.7. **VAL INT max.vector.number**. This positive 32 bit integer gives an upper bound on the total number of pixels representing intensities in the correlation plane of false targets

which the dc5 codes are attempting to suppress. This integer should be set to be 200 and probably no higher. Memory considerations may necessitate a smaller choice. However, if false.target.number = 0 and the boolean parameter average.numerator.mode (discussed in 10.1) is FALSE, then max.vector.number must be at least true.target.number.

6.8. **VAL INT n**. The 32 bit integer n is the number of columns in the complex array representing the spatial filter. The current version of the spatial filter design codes require that n be a power of 2 and that n be equal to m. The integer n must coincide with the integer filter.n used in the df codes. The notation may be unified in some future version of the tools.

6.9. **VAL INT number.of.processors**. This 32 bit integer represents the number of processors in the user's pipeline of transputer modules used by the dc5 codes. It must be greater than or equal to 2.

6.10. **VAL []BYTE rddc5dat.filename**. The linear byte array rddc5dat.filename is the name which is internal to the program for what we here call dc5.dat (discussed in 8.0). The dc5 program needs to know where to find dc5.dat at compilation time but not its contents. If the file dc5.dat is located in the \dc5\ subdirectory, then no pathname is needed. If the dc5.dat file is located elsewhere, the complete pathname must be included in rddc5dat.filename.

6.11. **VAL INT true.target.number**. This 32 bit integer is the total number of true targets in the training set. It must be at least 1.

6.12. **VAL INT target.number**. This 32 bit integer must always be set to equal (false.target.number +true.target.number).

7.0. **Creating the Bootable File dc5.btl**. One creates the bootable file dc5.btl by invoking the batch file dc5.bat, i.e., by typing "dc5". In order for this command to succeed the include files comp0.inc, dc5.inc, and dc5prot.inc, the library file dc5.lib, the linker indirect file dc5.lnk, the occam 2 text files boss5.occ and worker5.occ, and the text file dc5.pgm must

be in the \dc5\ subdirectory. A number of ancillary files are created by invoking dc5.bat. Do not be alarmed or confused. Only the final product, dc5.btl, is actually used.

8.0. **Editing dc5.dat**. The data file dc5.dat contains constants which are used in the dc5 program but which need not be specified at compile time. However, the location of the dc5.dat file itself must be known at compile time and is specified by rddc5dat.filename located in comp0.inc. The format of dc5.dat must be preserved. The user should make a backup copy before altering it. There is a large number of parameters which can be altered. Most of the time only a few need be changed. This large number of parameters gives the user a great deal of flexibility in placing input files into subdirectories and in controlling the length of time of the optimization.

8.1. **[80]BYTE correlation.plane.bool.mask.filename**. This linear byte array is the name for a bigm $\times$ bign file of booleans which the dc5 codes read into the two-dimensional array correlation.plane.bool.mask. The number of bytes, including the full pathname if necessary, must be no more than 80 characters. The dc5 codes permit a great deal of flexibility in suppressing signals in the correlation planes of the false targets in the training set. The (i,j)-th entry of correlation.plane.bool.mask has an entry which is TRUE if the corresponding entry in the correlation planes of the false targets are to be suppressed and which is FALSE otherwise. Note that if the (i,j)-th entry of correlation.plane.bool.mask is FALSE, then the corresponding entries in the correlation planes of the false targets are ignored in the computation of the SCR of the spatial filter. The correlation.plane.bool.mask is user defined. The user should set every entry of correlation.plane.bool.mask to be TRUE except in the most extraordinary instances. The array cpbm1.dat is supplied with the dc5 codes for the user's convenience. It contains a 256$\times$256 array of booleans, each of which is TRUE. Note that correlation.plane.bool.mask is cyclicly biased so that its (0,0)-th entry corresponds to the lowest spatial frequency in the correlation plane.

8.2. **[80]BYTE cpbm.key.filename**. This text file, needed by the dc5 spatial filter design codes, usually should be named c:\temp\cpbmkey.dat, for example. The df program generates a default version of this file. The user currently should always use this default

file. In the future more sophisticated user defined versions of this file might be of use. The number of characters in cpbm.key.filename cannot exceed 80.

8.3. **[80]BYTE filename.list.filename**. This linear array of bytes should be named filename.lis and kept in the \dc5\ subdirectory. The file filename.lis is supplied with this tool. It is discussed in some detail in the next subsection.

8.4. **[80]BYTE filter.bool.mask.filename**. This binary file is needed by the dc5 design codes. It usually should be named c:\temp\fbm.dat, for example. It is user defined during the first pass of the dc5 codes and is created by the fanal program for use during the second pass of the dc5 codes. The file fbm.dat is an m×n sized array of booleans (TRUE or FALSE) whose entries correspond in a one-to-one manner to each entry of the full complex spatial filter. For convenience call this spatial filter pof.t . If the (i,j)-th entry of fbm.dat is TRUE, then the fbm.dat file has no influence over the corresponding (i,j)-th entry in pof.tmp. If the (i,j)-th entry of fbm.dat is FALSE, then the corresponding (i.j)-th entry in pof.tmp is forced to be 0.0. It is very important to note that both pof.tmp and fbm.dat are arranged for two-dimensional FFT convenience. This means that pof.tmp is cyclicly biased in such a manner that the lowest spatial frequency contained in pof.tmp is located in the (0,0)-th entry. This file is discussed in more detail in df.doc. The number of characters in filter.bool.mask.filename, including its full pathname if necessary, cannot exceed 80.

8.5. **[80]BYTE pof.filename**. This linear byte array is the name of the file in which the filter.m×filter.n complex array containing the zero and continuous spatial filter is stored. The dc5 codes start with the initial guess for a zero and continuous phase-only filter, say \temp\pof.tmp, and iteratively generate improved versions of \temp\pof.tmp. The dc5 codes periodically write out updated versions of the spatial filter to \temp\pof.tmp. The number of characters in pof.filename, including its full pathname if necessary, cannot exceed 80.

8.6. **[80]BYTE scra.filename**. This is the name of a text filename written out to disk whenever pof.tmp is saved to the disk. It contains a wide variety of statistics on the

performance of the current version of pof.tmp. It should be named c:\temp\scra.tmp, for example. The number of characters in scra.filename, including its full pathname if necessary, cannot exceed 80.

8.7. **[80]BYTE target.filename**. This (often large) binary file, needed by the dc5 spatial filter design codes, usually should be named c:\temp\target.dat, for example. The df program generates this file. The number of characters in target.filename cannot exceed 80.

8.8. **[80]BYTE target.weight.filename**. This text file, needed by the dc5 spatial filter design codes, usually should be named c:\temp\tarwt.dat, for example. The df program generates a default version of this file. The user currently should always use this default file. In the future more sophisticated user defined versions of this file might be of use. The number of characters in tarwt.filename cannot exceed 80.

8.9. **REAL32 background**. List here the 32 bit floating point number representing the intensity of the background in which the targets are embedded. This number is first defined in df.dat. Since the df codes deal with byte images, each byte representing an intensity, the real number here should be between 0.0 and 255.0. Note that there is less flexibility here than in the df codes, which can handle different backgrounds for each new target set. Since this floating point number is only used in the documentation file \temp\scra.tmp and not in any computation, future versions of the dc5 codes may omit this parameter completely.

8.10. **INT image.stride**. List here the 32 bit integer which is the stride through the target input files first defined in df.dat. Note that there is less flexibility here than in the df codes, which can handle a different stride for each new target set. Since this integer is only used in the documentation file \temp\scra.tmp and not in any computation, future versions of the dc5 codes may omit this parameter completely.

8.11. **INT label.of.first.false.target.used**. This 32 bit integer is first defined in the df.dat. If there is no false target set or if there is more than one false target set or if there is a single false target set read in the image label input mode, set this 32 bit integer to 0. If there is a single false target set read in the sequential input mode, then set this 32 bit

integer to be the label of the very first image used in the false target file. Note that there is less flexibility here than in the df codes, which can handle different labels for each new target set. Since this integer is only used in the documentation file \temp\scra.tmp and not in any computation, future versions of the dc5 codes may omit this parameter completely.

8.12. **INT label.of.first.true.target.used**. This 32 bit integer is first defined and used in df.dat. If there is more than one true target set or if there is a single true target set but it is not read in the sequential input mode, set this 32 bit integer to 0. If there is a single true target set read in the sequential input mode, then set this 32 bit integer to be the label of the very first image used in the true target file. Note that there is less flexibility here than in the df codes, which can handle different labels for each new target set. Since this integer is only used in the documentation file \temp\scra.tmp and not in any computation, future versions of the dc5 codes may omit this parameter completely.

8.13. **[80]BYTE original.false.target.filename**. This linear byte array contains a list of the false target filenames, first defined in df.dat. They must be written on a single line no more than 80 characters long. For clarity the different names perhaps should be separated by commas or semicolons. Since this byte array is only used in the documentation file \temp\scra.tmp and not in any computation, future versions of the dc5 codes may omit this parameter completely.

8.14. **[80]BYTE original.true.target.filename**. This linear byte array contains a list of the true target filenames, first defined in df.dat. They must be written on a single line no more than 80 characters long. For clarity the different names perhaps should be separated by commas or semicolons. Since this byte array is only used in the documentation file \temp\scra.tmp and not in any computation, future versions of the dc5 codes may omit this parameter completely.

8.15. **[80]BYTE target.input.method**. The choices to be written here are "standard amplitude input method", "standard intensity input method", "intensity phase-encoded input method", or "amplitude phase-encoded input method". Which choice to use may be

gleaned from the appropriate booleans used in df.dat. Since this byte array is only used in the documentation file \temp\scra.tmp and not in any computation, future versions of the dc5 codes may omit this parameter completely.

8.16. **BOOL bool.normalize.filter**. This boolean is first used in the df codes and discussed in df.doc. It is first defined in df.dat and should have the same value here.

8.17. **BOOL bool.shift.mode**. This boolean is first used in the df codes and discussed in df.doc. It is first defined in df.dat and should have the same value here.

8.18. **BOOL bool.symmetrize.filter**. This boolean is first used in the df codes and discussed in df.doc. It is first defined in df.dat and should have the same value here.

8.19. **INT32 computation.time.upper.bound in seconds**. This 32 bit integer sets an upper bound in seconds on the length of the spatial filter design process. Once it is surpassed, the program will initiate an orderly shutdown process and will terminate.

8.20. **REAL32 scr.goal**. The 32 bit floating point number sets a goal for the SCR of the spatial filter during the design process. Once it is surpassed, the program will initiate an orderly shutdown process and will terminate. This goal may never be attained, but the program will terminate nonetheless in due course.

8.21. **INT total.passes.maximum**. This 32 bit integer sets a maximum on the total number of iterations, successful and unsuccessful, that the program will go through during the iterative optimization process used to design good zero and continuous phase spatial filters. Once it is surpassed, the program will initiate an orderly shutdown process and will terminate.

8.22. **BOOL bool.initial.save**. This boolean usually should be set to be TRUE. If TRUE, then the initial \temp\scra.tmp is saved to another file for more permanent storage using command.line.0 discussed in 2.23.

8.23. **[80]BYTE command.line.0**. Cf. 8.22. This command line should read "copy

c:\temp\scra.tmp c:\temp\poscra5a.tmp", for example. It is used to copy the first version of \temp\scra.tmp to another file for more permanent storage. The number of characters in this command line cannot exceed 80.

8.24. **[80]BYTE command.line.1**. This command line should read "copy c:\temp\scra.tmp c:\temp\poscra5b.tmp", for example. It is used to copy the final version of \temp\scra.tmp to another file for more permanent storage. The number of characters in this command line cannot exceed 80.

8.25. **[80]BYTE command.line.2**. This command line should read "copy c:\temp\pof.tmp c:\temp\pof5b.tmp", for example. It is used to copy the final version of \temp\pof.tmp to another file for more permanent storage. The number of characters in this command line cannot exceed 80.

9.0. **Editing filename.lis**. The following is a listing of the contents of the text file filename.lis:

–number.of.program.passes
9
misc0.dat
misc1.dat
misc2.dat
misc3.dat
misc4.dat
misc5.dat
misc6.dat
misc7.dat
miscend.dat

. The general format of this file must be preserved. The first line is a comment line. The second line tells the dc5 codes the number of the following files it will use. The last nine lines are the names of various files (potentially) used sequentially by the program. These files, discussed in the next subsection, contain various constants controlling aspects of the iterative

process used to optimize the design of zero and continuous phase spatial filters. The dc5 codes perform an iterative optimization controlled by a certain set of parameters, say those contained in misc0.dat. Then misc1.dat is read in and the iterative optimization process proceeds under the somewhat tighter control of the new parameters. Etc. At the end the dc5 codes make one final pass under the control of the parameters contained in miscend.dat, which results in fairly comprehensive statistics on the optimized zero and continuous phase spatial filter being written out to a file. The integer on the second line of filename.lis must be between 1 and 9. It may be changed using any text editor. If this integer is 1, then the lines of filename.lis must be permuted so that the third line of filename.lis is "miscend.dat". If this integer is 2, then the lines of filename.lis must be permuted so that the third line is misc1.dat and the fourth line is "miscend.dat". One edits filename.lis in similar fashion in the other cases. The larger the integer on the second line of this file, the finer the optimization.

10.0. **Editing the misc*.dat Files**. The user should be familiar with the discussion in subsection 9.0. The format of these files must be carefully preserved.

10.1. **BOOL average.numerator.mode**. This boolean, which of course assumes the values TRUE or FALSE, has an effect only if there are no false targets. In that instance there are two choices on how to proceed with the optimization. In general both should be tried, as there seems to be no general pattern as to which choice of TRUE or FALSE leads to better results. There seems to be some weak evidence that FALSE is the better choice when using binary imagery as inputs.

10.2. **INT choice.of.initial.z.vector**. This 32 bit integer should be set to 2.

10.3. **REAL32 clock.multiplier**. This 32 bit floating point number should be set to 64.0.

10.4. **REAL32 divisor1**. This 32 bit floating point number should be set to 1.0.

10.5. **REAL32 divisor.multiplier**. This 32 bit floating point number should be set to 1.6189034.

10.6. **REAL32 false.target.threshold**. These 32 bit floating point number should remain unaltered in the various misc*.dat files.

10.7. **REAL32 false.target.thresholded.number.fraction**. These 32 bit floating point numbers should remain unaltered in the various misc*.dat files.

10.8. **INT ftm.complete.recalculation.frequency**. This 32 bit integer should remain unaltered in the various misc*.dat files.

10.9. **INT iteration.number**. These 32 bit integers should remain unaltered in the various misc*.dat files.

10.10. **INT max.passes.since.last.success**. These 32 bit integers should remain unaltered in the various misc*.dat files.

10.11. **BOOL normalize.cor**. These booleans should remain unaltered in the various misc*.dat files.

10.12. **BOOL normalize.vector**. These booleans should remain unaltered in the various misc*.dat files.

10.13. **INT screen.write.out.frequency**. These 32 bit integers should be positive but must be 1 in miscend.dat.

10.14. **INT snra.write.out.frequency**. These 32 bit integers should be positive but must be 1 in miscend.dat.

10.15. **REAL32 true.target.threshold**. These 32 bit floating point numbers should remain unaltered in the various misc*.dat files.

10.16. **BOOL write.corcomp.timing**. This boolean should be set to be TRUE.

10.17. **BOOL write.denominator.timing**. This boolean should be set to be TRUE.

10.18. **BOOL write.numerator.timing**. This boolean should be set to be TRUE.

10.19. **BOOL write.vector.array.calculation.timing**. This boolean should be set to be TRUE.

10.20. **BOOL write.z.timing**. This boolean should be set to be TRUE.

11.0. **Running the Program**. If all is suitably prepared, run the dc5.btl program by invoking the simple batch file godc5.bat, i.e., type "godc5".

12.0. **Terminating the Program**. The program will terminate on its own, either successfully or in an erroneous state, in which case an error message will probably appear on the PC screen. The most common cause of an error message is the nonexistence of a file the program was seeking to use. The user can terminate the program by typing "control/break" and then typing "x".

13.0. **Cleaning up the \dc5\ subdirectory**. The \dc5\ subdirectory may be cleansed of the new files created by invoking the batch file cleanup.bat, i.e., by typing "cleanup". This is useful for housekeeping purposes. Be aware that dc5.btl is deleted in this process.

APPENDIX E

DC8.DOC

1.0. **Introduction and Purpose.** The purpose of this document is to give a very general overview of the dc8 (design code number 8) programs created by Robert R. Kallman. The dc8 codes start with the output of the first use of the dc5 codes, say \temp\filter.tmp, and iteratively generate improved versions of \temp\filter.tmp. The user should first read the file overview.doc for a view of where the dc8 programs fit into the general scheme of spatial filter design. The user should also be familiar with the contents of df.doc. Some of the notation used in overview.doc and in df.doc is also used here.

2.0. **Contents of dc8.zip**. The dc8 tools are usually supplied to the user in the binary file dc8.zip. This binary file is created by the data compression utility pkzip, version 2.04g. The following is a list of the files which will be created by applying the pkunzip utility to dc8.zip:

cleanup.bat
dc8.bat
godc8.bat
cpbm1.dat
dc8.dat
misc0.dat
misc1.dat
misc2.dat
misc3.dat
misc4.dat
misc5.dat
misc6.dat
misc7.dat
miscend.dat
dc8.doc

overview.doc

comp0.inc

dc8.inc

dc8prot.inc

dc8.lib

filename.lis

dc8.lnk

boss5.occ

worker5.occ

dc8.pgm

!!!WARNING!!! The user is advised to leave all of these files unaltered with the exceptions of comp0.dat and dc8.dat. Completely unpredictable, untoward, and erroneous results will be a consequence of not heeding this caveat.

3.0. **Overview of the dc8 Programs**. The dc8 programs are assumed to lie in the \dc8\ subdirectory. Unless stated otherwise all of the files listed in subsection 2.0 must lie in this subdirectory. The dc8 codes make use of the files \temp\target.dat, \temp\tarwt.dat, and \temp\cpbmkey.dat generated by the df codes. The dc8 codes start with the output of the first use of the dc5 codes, say \temp\filter.tmp, and iteratively generate improved versions of \temp\filter.tmp. The input file \temp\fbm.dat controls the choice of which pixels in \temp\filter.tmp are set equal to zero. The \temp\fbm.dat file is created either by the user and is discussed in more detail in df.doc. The \temp\fbm.dat file is discussed in more detail below. The zero pixels are not altered during the optimization. All other (nonzero) pixels are complex numbers of nonzero modulus less than or equal to one. The optimization proceeds by varying the phases and the amplitudes, constrained to be between zero and one, in an intelligent manner to drive the SCR for \temp\filter.tmp to as high a level as possible. When dc8 is complete, \temp\filter.tmp usually contains a wide distribution of amplitudes between zero and one in its entries, even though these amplitudes started with a value of either zero or one. Other ancillary files needed by the dc8 programs are discussed below.

4.0. **Hardware Requirements**. The dc8 programs must be used with a parallel computing system consisting of INMOS compatible transputer modules linked together in a linear pipeline. Transputers with a floating point processor are needed since the dc8 codes use certain instruction which are available on such transputers. Hence, the transputers should be a T800, T801, T805, or T9000 transputers. It is assumed that there are 8 megabytes of memory on the transputers. If not, minor changes must be made to the otherwise untouchable file dc8.pgm. For example, suppose the user has 4 megabytes of memory on the transputers of his system. Using any text editor change the line

SET processor (type, memsize :="T800", (8 TIMES M))

to

SET processor (type, memsize :="T800", (4 TIMES M)).

If the user's system has a mix of transputer modules with varying amounts of memory, then altering dc8.pgm is only slightly more complicated. The user should read the appropriate sections of the D7305A occam 2 toolset documentation for a detailed discussion of the syntax used in the hardware description contained in dc8.pgm.

The user should check the dc8.map file generated during compilation to make sure that there is sufficient memory on each of his modules to accommodate his computational requirements.

5.0. **Software Requirements**. This software tool must be used with the INMOS D7305A or compatible occam 2 toolset. This is necessary for trouble free compiling, linking, collecting, configuring, and running of the software tool.

6.0. **Editing comp0.inc**. The dc8 programs use three include files, comp0.inc, dc8.inc, and dc8prot.inc, all of which are listed in subsection 2.0. Two of these include files, dc8.inc and dc8prot.inc, must not be altered under any circumstances. The include file comp0.inc contains constants which must be known before compilation. For example, the language occam 2 needs to know all array sizes during compilation. This file must be in the \dc8\ subdirectory and its name must not be altered.

6.1. **VAL INT bigm**. The 32 bit integer bigm is the number of rows in the complex

array on which two-dimensional full complex fast Fourier transforms are performed. The current version of the dc8 programs require that bigm be a power of 2 and that bigm be the same as bign. Also, the current version of the spatial filter design codes restrict the value of bigm to be either m or 2m. The integer bigm must coincide with the integer ft.m used in the df codes. The notation may be unified in some future version of these tools.

6.2. **VAL INT bign**. The 32 bit integer bign is the number of rows in the complex array on which two-dimensional full complex fast Fourier transforms are performed. The current version of the dc8 programs require that bign be a power of 2 and that bign be the same as bigm. Also, the current version of the spatial filter design codes restrict the value of bign to be either n or 2n. The integer bign must coincide with the integer ft.n used in the df codes. The notation may be unified in some future version of these tools.

6.3. **VAL INT cpbm.number**. This constant has been largely used for experimental purposes. It should always be set to be 1.

6.4. **VAL INT false.target.number**. This 32 bit integer is the total number of false targets in the training set. As noted in overview.doc it should be set to 0 during the first use of the dc8 codes in a full blown spatial filter design.

6.5. **VAL INT m**. The 32 bit integer m is the number of rows in the complex array representing the spatial filter. The current version of the spatial filter design codes require that m be a power of 2 and that m be equal to n. The integer m must coincide with the integer filter.m used in the df codes. The notation may be unified in some future version of the tools.

6.6. **VAL INT max.number.of.program.passes**. This 32 bit integer should be kept at 9. It provides an upper bound on the integer appearing on the second line of filename.lis, discussed in 9.0.

6.7. **VAL INT max.vector.number**. This positive 32 bit integer gives an upper bound on the total number of pixels representing intensities in the correlation plane of false targets

which the dc8 codes are attempting to suppress. This integer should be set to be 200 and probably no higher. Memory considerations may necessitate a smaller choice. However, if false.target.number = 0 and the boolean parameter average.numerator.mode (discussed in 10.1) is FALSE, then max.vector.number must be at least true.target.number.

6.8. **VAL INT n**. The 32 bit integer n is the number of columns in the complex array representing the spatial filter. The current version of the spatial filter design codes require that n be a power of 2 and that n be equal to m. The integer n must coincide with the integer filter.n used in the df codes. The notation may be unified in some future version of the tools.

6.9. **VAL INT number.of.processors**. This 32 bit integer represents the number of processors in the user's pipeline of transputer modules used by the dc8 codes. It must be greater than or equal to 2.

6.10. **VAL []BYTE rddc8dat.filename**. The linear byte array rddc8dat.filename is the name which is internal to the program for what we here call dc8.dat (discussed in 8.0). The dc8 program needs to know where to find dc8.dat at compilation time but not its contents. If the file dc8.dat is located in the \dc8\ subdirectory, then no pathname is needed. If the dc8.dat file is located elsewhere, the complete pathname must be included in rddc8dat.filename.

6.11. **VAL INT true.target.number**. This 32 bit integer is the total number of true targets in the training set. It must be at least 1.

6.12. **VAL INT target.number**. This 32 bit integer must always be set to equal (false.target.number +true.target.number).

7.0. Creating the Bootable File dc8.btl. One creates the bootable file dc8.btl by invoking the batch file dc8.bat, i.e., by typing "dc8". In order for this command to succeed the include files comp0.inc, dc8.inc, and dc8prot.inc, the library file dc8.lib, the linker indirect file dc8.lnk, the occam 2 text files boss5.occ and worker5.occ, and the text file dc8.pgm must

be in the \dc8\ subdirectory. A number of ancillary files are created by invoking dc8.bat. Do not be alarmed or confused. Only the final product, dc8.btl, is actually used.

8.0. **Editing dc8.dat**. The data file dc8.dat contains constants which are used in the dc8 program but which need not be specified at compile time. However, the location of the dc8.dat file itself must be known at compile time and is specified by rddc8dat.filename located in comp0.inc. The format of dc8.dat must be preserved. The user should make a backup copy before altering it. There is a large number of parameters which can be altered. Most of the time only a few need be changed. This large number of parameters gives the user a great deal of flexibility in placing input files into subdirectories and in controlling the length of time of the optimization.

8.1. **[80]BYTE correlation.plane.bool.mask.filename**. This linear byte array is the name for a bigm $\times$ bign file of booleans which the dc8 codes read into the two-dimensional array correlation.plane.bool.mask. The number of bytes, including the full pathname if necessary, must be no more than 80 characters. The dc8 codes permit a great deal of flexibility in suppressing signals in the correlation planes of the false targets in the training set. The (i,j)-th entry of correlation.plane.bool.mask has an entry which is TRUE if the corresponding entry in the correlation planes of the false targets are to be suppressed and which is FALSE otherwise. Note that if the (i,j)-th entry of correlation.plane.bool.mask is FALSE, then the corresponding entries in the correlation planes of the false targets are ignored in the computation of the SCR of the spatial filter. The correlation.plane.bool.mask is user defined. The user should set every entry of correlation.plane.bool.mask to be TRUE except in the most extraordinary instances. The array cpbm1.dat is supplied with the dc8 codes for the user's convenience. It contains a 256$\times$256 array of booleans, each of which is TRUE. Note that correlation.plane.bool.mask is cyclicly biased so that its (0,0)-th entry corresponds to the lowest spatial frequency in the correlation plane.

8.2. **[80]BYTE cpbm.key.filename**. This text file, needed by the dc8 spatial filter design codes, usually should be named c:\temp\cpbmkey.dat, for example. The df program generates a default version of this file. The user currently should always use this default

file. In the future more sophisticated user defined versions of this file might be of use. The number of characters in cpbm.key.filename cannot exceed 80.

8.3. **[80]BYTE filename.list.filename**. This linear array of bytes should be named filename.lis and kept in the \dc8\ subdirectory. The file filename.lis is supplied with this tool. It is discussed in some detail in the next subsection.

8.4. **[80]BYTE filter.bool.mask.filename**. This binary file is needed by the dc8 design codes. It usually should be named c:\temp\fbm.dat, for example. It is user defined. The file fbm.dat is an m×n sized array of booleans (TRUE or FALSE) whose entries correspond in a one-to-one manner to each entry of the full complex spatial filter. For convenience call this spatial filter filter.tmp. If the (i,j)-th entry of fbm.dat is TRUE, then the fbm.dat file has no influence over the corresponding (i,j)-th entry in filter.tmp. If the (i,j)-th entry of fbm.dat is FALSE, then the corresponding (i,j)-th entry in filter.tmp is forced to be 0.0. It is very important to note that both filter.tmp and fbm.dat are arranged for two-dimensional FFT convenience. This means that filter.tmp is cyclicly biased in such a manner that the lowest spatial frequency contained in filter.tmp is located in the (0,0)-th entry. This file is discussed in more detail in df.doc. The number of characters in filter.bool.mask.filename, including its full pathname if necessary, cannot exceed 80.

8.5. **[80]BYTE filter.filename**. This linear byte array is the name of the file in which the filter.m×filter.n complex array containing the general spatial filter is stored. The dc8 codes start with the output of the first use of the dc5 codes, say \temp\filter.tmp, and iteratively generate improved versions of \temp\filter.tmp. The dc8 codes periodically write out updated versions of the spatial filter to \temp\filter.tmp. The number of characters in filter.filename, including its full pathname if necessary, cannot exceed 80.

8.6. **[80]BYTE scra.filename**. This is the name of a text filename written out to disk whenever filter.tmp is saved to the disk. It contains a wide variety of statistics on the performance of the current version of filter.tmp. It should be named c:\temp\scra.tmp, for example. The number of characters in scra.filename, including its full pathname if necessary,

cannot exceed 80.

8.7. **[80]BYTE target.filename**. This (often large) binary file, needed by the dc8 spatial filter design codes, usually should be named c:\temp\target.dat, for example. The df program generates this file. The number of characters in target.filename cannot exceed 80.

8.8. **[80]BYTE target.weight.filename**. This text file, needed by the dc8 spatial filter design codes, usually should be named c:\temp\tarwt.dat, for example. The df program generates a default version of this file. The user currently should always use this default file. In the future more sophisticated user defined versions of this file might be of use. The number of characters in tarwt.filename cannot exceed 80.

8.9. **INT gradient.count.threshold**. This 32 bit positive integer parameter is reserved for experimental use. It should be set to 10.

8.10. **REAL32 background**. List here the 32 bit floating point number representing the intensity of the background in which the targets are embedded. This number is first defined in df.dat. Since the df codes deal with byte images, each byte representing an intensity, the real number here should be between 0.0 and 255.0. Note that there is less flexibility here than in the df codes, which can handle different backgrounds for each new target set. Since this floating point number is only used in the documentation file \temp\scra.tmp and not in any computation, future versions of the dc8 codes may omit this parameter completely.

8.11. **INT image.stride**. List here the 32 bit integer which is the stride through the target input files first defined in df.dat. Note that there is less flexibility here than in the df codes, which can handle a different stride for each new target set. Since this integer is only used in the documentation file \temp\scra.tmp and not in any computation, future versions of the dc8 codes may omit this parameter completely.

8.12. **INT label.of.first.false.target.used**. This 32 bit integer is first defined in the df.dat. If there is no false target set or if there is more than one false target set or if there is a single false target set read in the image label input mode, set this 32 bit integer to 0.

If there is a single false target set read in the sequential input mode, then set this 32 bit integer to be the label of the very first image used in the false target file. Note that there is less flexibility here than in the df codes, which can handle different labels for each new target set. Since this integer is only used in the documentation file \temp\scra.tmp and not in any computation, future versions of the dc8 codes may omit this parameter completely.

8.13. **INT label.of.first.true.target.used**. This 32 bit integer is first defined and used in df.dat. If there is more than one true target set or if there is a single true target set but it is not read in the sequential input mode, set this 32 bit integer to 0. If there is a single true target set read in the sequential input mode, then set this 32 bit integer to be the label of the very first image used in the true target file. Note that there is less flexibility here than in the df codes, which can handle different labels for each new target set. Since this integer is only used in the documentation file \temp\scra.tmp and not in any computation, future versions of the dc8 codes may omit this parameter completely.

8.14. **[80]BYTE original.false.target.filename**. This linear byte array contains a list of the false target filenames, first defined in df.dat. They must be written on a single line no more than 80 characters long. For clarity the different names perhaps should be separated by commas or semicolons. Since this byte array is only used in the documentation file \temp\scra.tmp and not in any computation, future versions of the dc8 codes may omit this parameter completely.

8.15. **[80]BYTE original.true.target.filename**. This linear byte array contains a list of the true target filenames, first defined in df.dat. They must be written on a single line no more than 80 characters long. For clarity the different names perhaps should be separated by commas or semicolons. Since this byte array is only used in the documentation file \temp\scra.tmp and not in any computation, future versions of the dc8 codes may omit this parameter completely.

8.16. **[80]BYTE target.input.method**. The choices to be written here are "standard amplitude input method", "standard intensity input method", "intensity phase-encoded in-

put method", or "amplitude phase-encoded input method". Which choice to use may be gleaned from the appropriate booleans used in df.dat. Since this byte array is only used in the documentation file \temp\scra.tmp and not in any computation, future versions of the dc8 codes may omit this parameter completely.

8.17. **BOOL bool.normalize.filter**. This boolean is first used in the df codes and discussed in df.doc. It is first defined in df.dat and should have the same value here.

8.18. **BOOL bool.shift.mode**. This boolean is first used in the df codes and discussed in df.doc. It is first defined in df.dat and should have the same value here.

8.19. **BOOL bool.symmetrize.filter**. This boolean is first used in the df codes and discussed in df.doc. It is first defined in df.dat and should have the same value here.

8.20. **INT32 computation.time.upper.bound in seconds**. This 32 bit integer sets an upper bound in seconds on the length of the spatial filter design process. Once it is surpassed, the program will initiate an orderly shutdown process and will terminate.

8.21. **REAL32 scr.goal**. The 32 bit floating point number sets a goal for the SCR of the spatial filter during the design process. Once it is surpassed, the program will initiate an orderly shutdown process and will terminate. This goal may never be attained, but the program will terminate nonetheless in due course.

8.22. **INT total.passes.maximum**. This 32 bit integer sets a maximum on the total number of iterations, successful and unsuccessful, that the program will go through during the iterative optimization process used to design good zero and continuous phase spatial filters. Once it is surpassed, the program will initiate an orderly shutdown process and will terminate.

8.23. **BOOL bool.initial.save**. This boolean usually should be set to be TRUE. If TRUE, then the initial \temp\scra.tmp is saved to another file for more permanent storage using command.line.0 discussed in 2.23.

8.24. **[80]BYTE command.line.0**. Cf. 8.22. This command line should read "copy c:\temp\scra.tmp c:\temp\poscra8a.tmp", for example. It is used to copy the first version of \temp\scra.tmp to another file for more permanent storage. The number of characters in this command line cannot exceed 80.

8.25. **[80]BYTE command.line.1**. This command line should read "copy c:\temp\scra.tmp c:\temp\poscra8b.tmp", for example. It is used to copy the final version of \temp\scra.tmp to another file for more permanent storage. The number of characters in this command line cannot exceed 80.

8.26. **[80]BYTE command.line.2**. This command line should read "copy c:\temp\filter.tmp c:\temp\fil8b.tmp", for example. It is used to copy the final version of \temp\filter.tmp to another file for more permanent storage. The number of characters in this command line cannot exceed 80.

9.0. **Editing filename.lis**. The following is a listing of the contents of the text file filename.lis:

–number.of.program.passes
9
misc0.dat
misc1.dat
misc2.dat
misc3.dat
misc4.dat
misc5.dat
misc6.dat
misc7.dat
miscend.dat

. The general format of this file must be preserved. The first line is a comment line. The second line tells the dc8 codes the number of the following files it will use. The last nine lines are the names of various files (potentially) used sequentially by the program. These files,

discussed in the next subsection, contain various constants controlling aspects of the iterative process used to optimize the design of zero and continuous phase spatial filters. The dc8 codes perform an iterative optimization controlled by a certain set of parameters, say those contained in misc0.dat. Then misc1.dat is read in and the iterative optimization process proceeds under the somewhat tighter control of the new parameters. Etc. At the end the dc8 codes make one final pass under the control of the parameters contained in miscend.dat, which results in fairly comprehensive statistics on the optimized zero and continuous phase spatial filter being written out to a file. The integer on the second line of filename.lis must be between 1 and 9. It may be changed using any text editor. If this integer is 1, then the lines of filename.lis must be permuted so that the third line of filename.lis is "miscend.dat". If this integer is 2, then the lines of filename.lis must be permuted so that the third line is misc1.dat and the fourth line is "miscend.dat". One edits filename.lis in similar fashion in the other cases. The larger the integer on the second line of this file, the finer the optimization.

10.0. Editing the misc*.dat Files. The user should be familiar with the discussion in subsection 9.0. The format of these files must be carefully preserved.

10.1. **BOOL average.numerator.mode**. This boolean, which of course assumes the values TRUE or FALSE, has an effect only if there are no false targets. In that instance there are two choices on how to proceed with the optimization. In general both should be tried, as there seems to be no general pattern as to which choice of TRUE or FALSE leads to better results. There seems to be some weak evidence that FALSE is the better choice when using binary imagery as inputs.

10.2. **INT choice.of.initial.z.vector**. This 32 bit integer should be set to 2.

10.3. **REAL32 clock.multiplier**. This 32 bit floating point number should be set to 64.0.

10.4. **REAL32 divisor1**. This 32 bit floating point number should be set to 1.0.

10.5. **REAL32 divisor.multiplier**. This 32 bit floating point number should be set to

1.6189034.

10.6. **REAL32 false.target.threshold**. These 32 bit floating point number should remain unaltered in the various misc*.dat files.

10.7. **REAL32 false.target.thresholded.number.fraction**. These 32 bit floating point numbers should remain unaltered in the various misc*.dat files.

10.8. **INT ftm.complete.recalculation.frequency**. This 32 bit integer should remain unaltered in the various misc*.dat files.

10.9. **INT iteration.number**. These 32 bit integers should remain unaltered in the various misc*.dat files.

10.10. **INT max.passes.since.last.success**. These 32 bit integers should remain unaltered in the various misc*.dat files.

10.11. **BOOL normalize.cor**. These booleans should remain unaltered in the various misc*.dat files.

10.12. **BOOL normalize.vector**. These booleans should remain unaltered in the various misc*.dat files.

10.13. **INT screen.write.out.frequency**. These 32 bit integers should be positive but must be 1 in miscend.dat.

10.14. **INT smra.write.out.frequency**. These 32 bit integers should be positive but must be 1 in miscend.dat.

10.15. **REAL32 true.target.threshold**. These 32 bit floating point numbers should remain unaltered in the various misc*.dat files.

10.16. **BOOL write.corcomp.timing**. This boolean should be set to be TRUE.

10.17. **BOOL write.denominator.timing**. This boolean should be set to be TRUE.

10.18. **BOOL write.numerator.timing**. This boolean should be set to be TRUE.

10.19. **BOOL write.vector.array.calculation.timing**. This boolean should be set to be TRUE.

10.20. **BOOL write.z.timing**. This boolean should be set to be TRUE.

11.0. **Running the Program**. If all is suitably prepared, run the dc8.btl program by invoking the simple batch file godc8.bat, i.e., type "godc8".

12.0. **Terminating the Program**. The program will terminate on its own, either successfully or in an erroneous state, in which case an error message will probably appear on the PC screen. The most common cause of an error message is the nonexistence of a file the program was seeking to use. The user can terminate the program by typing "control/break" and then typing "x".

13.0. **Cleaning up the \dc8\ subdirectory**. The \dc8\ subdirectory may be cleansed of the new files created by invoking the batch file cleanup.bat, i.e., by typing "cleanup". This is useful for housekeeping purposes. Be aware that dc8.btl is deleted in this process.

APPENDIX F

FANAL.DOC

1.0. **Introduction and Purpose**. The purpose of this document is to give a very general overview of the fanal (filter analysis) program created by Robert R. Kallman for analyzing a general spatial filter and deriving a variety of ancillary arrays from it. The user should first read the file overview.doc for a view of where the fanal program fits into the general scheme of filter design. Some of the notation used in overview.doc is also used here.

2.0. **Contents of fanal.zip**. The fanal tools are usually supplied to the user in the binary file fanal.zip. This binary file is created by the data compression utility pkzip, version 2.04g. The following is a list of the files which will be created by applying the pkunzip utility to fanal.zip:

fanal.bat

cleanup.bat

gofan.bat

fanal.dat

fanal.doc

overview.doc

fanal.inc

fanal.lib

fanal.lnk

fanal.occ

fanal.pgm

!!!WARNING!!! The user is advised to leave all of these files unaltered with the exceptions of fanal.dat and fanal.inc. Completely unpredictable, untoward, and erroneous results will be a likely consequence of not heeding this caveat.

3.0. **Overview of the fanal Program**. The fanal program is assumed to lie in the \fanal\ subdirectory. Unless stated otherwise all of the files listed in subsection 2.0 must lie in this subdirectory. The use of this program usually only makes sense with

the output \temp\filter.tmp of the dc8 codes. The reader should familiarize himself with dc8.doc before proceeding further. The fanal program starts with \temp\filter.tmp and a user defined threshold and outputs three files: \temp\fbm001.tmp, \temp\fil001.tmp, and \temp\pof001.tmp. The file \temp\fbm001.tmp, an array of booleans, has entry which is FALSE if the corresponding entry in \temp\filter.tmp has an amplitude less than or equal to the threshold and has an entry which is TRUE otherwise. Note that if an entry of \temp\fbm.dat is FALSE then the corresponding entry of \temp\fbm001.tmp is FALSE. The file \temp\fil001.tmp is a spatial filter created from \temp\filter.tmp by setting an entry of \temp\fil001.tmp equal to zero if the corresponding entry of \temp\fbm001.tmp is FALSE and by setting an entry of \temp\fil001.tmp equal to the corresponding entry in \temp\filter.tmp otherwise. The file \temp\fil001.tmp will be of no use to us in this discussion. The file \temp\pof001.tmp, a zero and continuous phase filter, is obtained from \temp\filter.tmp by setting an entry of \temp\pof001.tmp equal to zero if the corresponding entry of \temp\fbm001.tmp is FALSE and by setting an entry of \temp\pof001.tmp equal to the phase of the corresponding entry of \temp\filter.tmp otherwise.

4.0. **Hardware Requirements**. The fanal program must be used with an INMOS compatible transputer module. A transputer with a floating point processor is recommended for efficient operation. Hence, the transputer should be a T800, T801, T805, or T9000 transputer. It is assumed that there are 8 megabytes of memory on the transputer. If not, a minor change must be made to the otherwise untouchable file fanal.pgm. For example, suppose the user has 4 megabytes of memory on the root transputer of his system. Using any text editor change the line

SET processor (type, memsize :="T800", (8 TIMES M))

to

SET processor (type, memsize :="T800", (4 TIMES M)).

The user should read the appropriate sections of the D7305A occam 2 toolset documentation for a detailed discussion of the syntax used in the hardware description contained in fanal.pgm.

The user should check the fanal.map file generated during compilation to make sure

that there is sufficient memory on each of his modules to accommodate his computational requirements.

5.0. **Software Requirements.** This software tool must be used with the INMOS D7305A or compatible occam 2 toolset. This is necessary for trouble free compiling, linking, collecting, configuring, and running of the software tool.

6.0. **Editing fanal.inc.** The include file fanal.inc contains constants which must be known before compilation. For example, the language occam 2 needs to know all array sizes during compilation. This file must be in the \fanal\ subdirectory and its name must not be altered.

6.1. **VAL INT m.** The 32 bit integer m is the number of rows in the complex array \temp\filter.tmp.

6.2. **VAL INT n.** The 32 bit integer n is the number of columns in \temp\filter.tmp.

6.3. **VAL []BYTE rdfandat.filename.** The linear byte array rdfandat.filename is the name which is internal to the program for what we here call fanal.dat (discussed in 8.0). The fanal program needs to know where to find fanal.dat at compilation time but not its contents. If the file fanal.dat is located in the \fanal\ subdirectory, then no pathname is needed. If the fanal.dat file is located elsewhere, the complete pathname must be included in rdfandat.filename.

7.0. **Creating the Bootable File fanal.btl.** One creates the bootable file fanal.btl by invoking the batch file fanal.bat, i.e., by typing "fanal". In order for this command to succeed the include file fanal.inc, the library file fanal.lib, the linker indirect file fanal.lnk, the occam 2 text file fanal.occ, and the text file fanal.pgm must be in the \fanal\ subdirectory. A number of ancillary files are created by invoking fanal.bat. Do not be alarmed or confused. Only the final product, fanal.btl, is actually used.

8.0. **Editing fanal.dat.** The data file fanal.dat contains constants which are used in

the fanal program but which need not be specified at compile time. However, the location of the fanal.dat file itself must be known at compile time and is specified by rdfandat.filename located in fanal.inc. The format of fanal.dat must be preserved. The user should make a backup copy before altering it.

8.1. **REAL32 desired.zeroed.fraction**. The IEEE 32 bit real number desired.zeroed.fraction must be between 0.0 and 1.0. It specifies which fraction of the pixels in \temp\filter.tmp are to be unalterably fixed to be zero. The pixels chosen are always those with the smallest amplitudes.

8.2. **[80]BYTE complex.input.filename**. The linear byte array [80]BYTE complex.input.filename is self descriptive. It contains the name of the input file to be processed (cf. 3.0). If this file lies in the \fanal\ subdirectory, then only its name need be given. Otherwise the complete pathname, e.g., c:\temp\filter.tmp, must be given. Note that the total number of characters used must be less than or equal to 80.

8.3. **[80]BYTE boolean.output.filename**. The linear byte array [80]BYTE boolean.output.filename is self descriptive. It contains the name of output boolean file (cf. 3.0). If this file is to lie in the \fanal\ subdirectory, then only its name need be given. Otherwise the complete pathname, e.g., c:\temp\fbm001.tmp, must be given. Note that the total number of characters used must be less than or equal to 80.

8.4. **[80]BYTE complex.output.filename**.The linear byte array [80]BYTE complex.output.filename is self descriptive. It contains the name of output complex file which contains the truncated version of filter.tmp (cf. 3.0). If this file is to lie in the \fanal\ subdirectory, then only its name need be given. Otherwise the complete pathname, e.g., c:\temp\fil001.tmp, must be given. Note that the total number of characters used must be less than or equal to 80.

8.5. **[80]BYTE pof.output.filename**. The linear byte array [80]BYTE pof.output.filename is self descriptive. It contains the name of output complex file which will contains the zero and continuous phase filter (cf. 3.0). If this file is to lie in the \fanal\

subdirectory, then only its name need be given. Otherwise the complete pathname, e.g., c:\temp\pof001.tmp, must be given. Note that the total number of characters used must be less than or equal to 80.

8.6. **REAL32 amplitude.cutoff.fraction**. The IEEE 32 bit real number amplitude.cutoff.fraction should always be set to 1.0.

8.7. **INT maximum.number.of.iterations**. The fanal program goes through an iterative process to compute its output files. This iterative process will fail for a very some peculiar input files. The 32 bit integer maximum.number of iterations sets an upper bound on the number of iterations in these cases and the program will terminate in a timely manner. The number should be at least $[\log_2(m \cdot n)]$.

8.8. **REAL32 fraction.tolerance**. The IEEE 32 bit real number fraction.tolerance should always be greater than $1/(m \cdot n)$. Once fixed for a choice of m and n, it should remain unaltered.

8.9. **REAL32 threshold.tolerance**. The IEEE 32 bit real number fraction.tolerance should always be at least $1/(m \cdot n)$. Once fixed for a choice of m and n, it should remain unaltered.

9.0. **Running the Program**. If all is suitably prepared, run the fanal.btl program by invoking the simple batch file gofan.bat, i.e., type "gofan".

10.0. **Terminating the Program**. The program will terminate on its own, either successfully or in an erroneous state, in which case an error message will probably appear on the PC screen. The most common cause of an error message is the nonexistence of a file the program was seeking to use. The user can terminate the program by typing "control/break" and then typing "x".

11.0. **Cleaning up the \fanal\ Subdirectory**. The \fanal\ subdirectory may be cleansed of the new files created by invoking the batch file cleanup.bat, i.e., by typing

"cleanup". This is useful for housekeeping purposes. Be aware that fanal.btl is deleted in this process.

APPENDIX G

DC5Z.DOC

1.0. **Introduction and Purpose.** The purpose of this document is to give a very general overview of the dc5z (design code number 5z) programs created by Robert R. Kallman. The dc5z codes start with the a good zero and continuous phase-only filter, say \temp\pof.tmp, and optimally discretize it into a zero and user selected finite number of equally spaced phase states, say \temp\pod.tmp. The user should first read the file overview.doc for a view of where the dc5z programs fit into the general scheme of spatial filter design. The user should also be familiar with the contents of df.doc. Some of the notation used in overview.doc and df.doc is also used here.

2.0. **Contents of dc5z.zip.** The dc5z tools are usually supplied to the user in the binary file dc5z.zip. This binary file is created by the data compression utility pkzip, version 2.04g. The following is a list of the files which will be created by applying the pkunzip utility to dc5z.zip:

cleanup.bat
dc5z.bat
godc5z.bat
cpbm1.dat
dc5z.dat
zmisc15.dat
zmisc16.dat
zmisc2.dat
dc5z.doc
overview.doc
comp0.inc
dc5z.inc
dc5zprot.inc
dc5z.lib

dc5z.lnk

boss5z.occ

worker5z.occ

dc5z.pgm

!!!WARNING!!! The user is advised to leave all of these files unaltered with the exceptions of comp0.dat and dc5z.dat. Completely unpredictable, untoward, and erroneous results will be a consequence of not heeding this caveat.

3.0. **Overview of the dc5z Programs**. The dc5z programs are assumed to lie in the \dc5z\ subdirectory. Unless stated otherwise all of the files listed in subsection 2.0 must lie in this subdirectory. The dc5z codes make use of the files \temp\target.dat, \temp\tarwt.dat, and \temp\cpbmkey.dat generated by the df codes. The dc5z codes optimally discretize \temp\pof.tmp, usually the output of the second use of the dc5 codes, into a filter whose entries are zero and a finite number of equally spaced phase states. The number of equally spaced phase states is user selectable. The output of the dc5z codes can be found in the file \temp\pod.tmp. The input file \temp\fbm.dat controls the choice of which pixels in \temp\pod.tmp are set equal to zero. The \temp\fbm.dat file usually is created by the fanal program. The \temp\fbm.dat file is discussed in more detail below. The zero pixels are not altered during the optimization. All other (nonzero) pixels are nth roots of unity, where n is user selected. The optimal discretization proceeds by driving the signal-to-clutter ratio (SCR) for the discrete filter to as high a level as possible. Other ancillary files needed by the dc5z programs are discussed below.

4.0. **Hardware Requirements**. The dc5z programs must be used with a parallel computing system consisting of INMOS compatible transputer modules linked together in a linear pipeline. Transputers with a floating point processor are needed since the dc5z codes use certain instruction which are available on such transputers. Hence, the transputers should be a T800, T801, T805, or T9000 transputers. It is assumed that there are 8 megabytes of memory on the transputers. If not, minor changes must be made to the otherwise untouchable file dc5z.pgm. For example, suppose the user has 4 megabytes of memory on the transputers of his system. Using any text editor change the line

SET processor (type, memsize :="T800", (8 TIMES M))

to

SET processor (type, memsize :="T800", (4 TIMES M)).

If the user's system has a mix of transputer modules with varying amounts of memory, then altering dc5z.pgm is only slightly more complicated. The user should read the appropriate sections of the D7305A occam 2 toolset documentation for a detailed discussion of the syntax used in the hardware description contained in dc5z.pgm.

The user should check the dc5z.map file generated during compilation to make sure that there is sufficient memory on each of his modules to accommodate his computational requirements.

5.0. **Software Requirements**. This software tool must be used with the INMOS D7305A or compatible occam 2 toolset. This is necessary for trouble free compiling, linking, collecting, configuring, and running of the software tool.

6.0. **Editing comp0.inc**. The dc5z programs use three include files, comp0.inc, dc5z.inc, and dc5zprot.inc, all of which are listed in subsection 2.0. Two of these include files, dc5z.inc and dc5zprot.inc, must not be altered under any circumstances. The include file comp0.inc contains constants which must be known before compilation. For example, the language occam 2 needs to know all array sizes during compilation. This file must be in the \dc5z\ subdirectory and its name must not be altered.

6.1. **VAL INT bigm**. The 32 bit integer bigm is the number of rows in the complex array on which two-dimensional full complex fast Fourier transforms are performed. The current version of the dc5z programs require that bigm be a power of 2 and that bigm be the same as bign. Also, the current version of the spatial filter design codes restrict the value of bigm to be either m or 2m. The integer bigm must coincide with the integer of the same name used in the dc5 codes.

6.2. **VAL INT bign**. The 32 bit integer bign is the number of rows in the complex array on which two-dimensional full complex fast Fourier transforms are performed. The current

version of the dc5z programs require that bign be a power of 2 and that bign be the same as bigm. Also, the current version of the spatial filter design codes restrict the value of bign to be either n or 2n. The integer bign must coincide with the integer of the same name used in the dc5 codes.

6.3. **VAL INT cpbm.number**. This constant has been largely used for experimental purposes. It should always be set to be 1.

6.4. **VAL INT false.target.number**. This 32 bit integer is the total number of false targets in the training set. As noted in overview.doc it should be set to 0 during the first use of the dc5z codes in a full blown spatial filter design.

6.5. **VAL INT m**. The 32 bit integer m is the number of rows in the complex array representing the spatial filter. The current version of the spatial filter design codes require that m be a power of 2 and that m be equal to n. The integer m must coincide with the integer of the same name used in the dc5 codes.

6.6. **VAL INT n**. The 32 bit integer n is the number of columns in the complex array representing the spatial filter. The current version of the spatial filter design codes require that n be a power of 2 and that n be equal to m. The integer m must coincide with the integer of the same name used in the dc5 codes.

6.7. **VAL INT number.of.processors**. This 32 bit integer represents the number of processors in the user's pipeline of transputer modules used by the dc5z codes. It must be greater than or equal to 2.

6.8. **VAL []BYTE rddc5zdat.filename**. The linear byte array rddc5zdat.filename is the name which is internal to the program for what we here call dc5z.dat (discussed in 8.0). The dc5z program needs to know where to find dc5z.dat at compilation time but not its contents. If the file dc5z.dat is located in the \dc5z\ subdirectory, then no pathname is needed. If the dc5z.dat file is located elsewhere, the complete pathname must be included in rddc5zdat.filename.

6.9. **VAL INT true.target.number**. This 32 bit integer is the total number of true targets in the training set. It must be at least 1.

6.10. **VAL INT target.number**. This 32 bit integer must always be set to equal (false.target.number +true.target.number).

7.0. **Creating the Bootable File dc5z.btl**. One creates the bootable file dc5z.btl by invoking the batch file dc5z.bat, i.e., by typing "dc5z". In order for this command to succeed the include files comp0.inc, dc5z.inc, and dc5zprot.inc, the library file dc5z.lib, the linker indirect file dc5z.lnk, the occam 2 text files boss5z.occ and worker5z.occ, and the text file dc5z.pgm must be in the \dc5z\ subdirectory. A number of ancillary files are created by invoking dc5z.bat. Do not be alarmed or confused. Only the final product, dc5z.btl, is actually used.

8.0. **Editing dc5z.dat**. The data file dc5z.dat contains constants which are used in the dc5z program but which need not be specified at compile time. However, the location of the dc5z.dat file itself must be known at compile time and is specified by rddc5zdat.filename located in comp0.inc. The format of dc5z.dat must be preserved. The user should make a backup copy before altering it. There is a large number of parameters which can be altered. Most of the time only a few need be changed. This large number of parameters gives the user a great deal of flexibility in placing input files into subdirectories the fineness of the discretization.

8.1. **[80]BYTE correlation.plane.bool.mask.filename**. This linear byte array is the name for a bigm × bign file of booleans which the dc5z codes read into the two-dimensional array correlation.plane.bool.mask. The number of bytes, including the full pathname if necessary, must be no more than 80 characters. The dc5z codes permit a great deal of flexibility in suppressing signals in the correlation planes of the false targets in the training set. The (i,j)-th entry of correlation.plane.bool.mask has an entry which is TRUE if the corresponding entry in the correlation planes of the false targets are to be suppressed and which is FALSE otherwise. Note that if the (i,j)-th entry of correlation.plane.bool.mask

is FALSE, then the corresponding entries in the correlation planes of the false targets are ignored in the computation of the SCR of the spatial filter. The correlation.plane.bool.mask is user defined. The user should set every entry of correlation.plane.bool.mask to be TRUE except in the most extraordinary instances. The array cpbm1.dat is supplied with the dc5z codes for the user's convenience. It contains a 256×256 array of booleans, each of which is TRUE. Note that correlation.plane.bool.mask is cyclicly biased so that its (0,0)-th entry corresponds to the lowest spatial frequency in the correlation plane.

8.2. **[80]BYTE cpbm.key.filename**. This text file, needed by the dc5z spatial filter design codes, usually should be named c:\temp\cpbmkey.dat, for example. The df program generates a default version of this file. The user currently should always use this default file. In the future more sophisticated user defined versions of this file might be of use. The number of characters in cpbm.key.filename cannot exceed 80.

8.3. **[80]BYTE filter.bool.mask.filename**. This binary file is needed by the dc5z design codes. It usually should be named c:\temp\fbm.dat, for example. It may be user defined but is usually created by the fanal program. The file fbm.dat is an m×n sized array of booleans (TRUE or FALSE) whose entries correspond in a one-to-one manner to each entry of \temp\pof.tmp. If the (i,j)-th entry of fbm.dat is TRUE, then the fbm.dat file has no influence over the corresponding (i,j)-th entry in pof.tmp. If the (i,j)-th entry of fbm.dat is FALSE, then the corresponding (i,j)-th entry in pof.tmp is forced to be 0.0. It is very important to note that both pof.tmp and fbm.dat are arranged for two-dimensional FFT convenience. This means that pof.tmp is cyclicly biased in such a manner that the lowest spatial frequency contained in pof.tmp is located in the (0,0)-th entry. This file is discussed in more detail in df.doc. The number of characters in filter.bool.mask.filename, including its full pathname if necessary, cannot exceed 80.

8.4. **[80]BYTE pod.filename**. This linear byte array is the name of the file in which the filter.m × filter.n complex array containing the zero and equally spaced discrete phase spatial filter is stored. It should be named c:\temp\pod.tmp, for example. The dc5z codes start with a good zero and continuous phase-only filter, say \temp\pof.tmp, and intelligently

searches for an optimal discretization so that the resulting zero and discrete phase state filter has highest possible SCR. The dc5z codes write out an updated version of the discrete spatial filter to \temp\pod.tmp with every improvement in SCR. The number of characters in pod.filename, including its full pathname if necessary, cannot exceed 80.

8.5. **[80]BYTE pof.filename**. This linear byte array is the name of the file in which the filter.m × filter.n complex array containing the input zero and continuous spatial filter is stored. It should be named c:\temp\pof.tmp, for example. The number of characters in pof.filename, including its full pathname if necessary, cannot exceed 80.

8.6. **[80]BYTE scr.filename**. This is the name of a text filename written out to disk whenever pod.tmp is saved to the disk. It contains a wide variety of statistics on the performance of the current version of pod.tmp. It should be named c:\temp\scr.tmp, for example. The number of characters in scr.filename, including its full pathname if necessary, cannot exceed 80.

8.7. **[80]BYTE target.filename**. This (often large) binary file, needed by the dc5z spatial filter design codes, usually should be named c:\temp\target.dat, for example. The df program generates this file. The number of characters in target.filename cannot exceed 80.

8.8. **[80]BYTE target.weight.filename**. This text file, needed by the dc5z spatial filter design codes, usually should be named c:\temp\tarwt.dat, for example. The df program generates a default version of this file. The user currently should always use this default file. In the future more sophisticated user defined versions of this file might be of use. The number of characters in tarwt.filename, including its full pathname if necessary, cannot exceed 80.

8.9. **[80]BYTE zmisc.filename**. This file, discussed in the next subsection, contains various constants controlling aspects of the optimal discretization process. It should be named zmisc2.dat, for example, if a zero and two phase filter filter is to be created. The number of characters in zmisc.filename, including its full pathname if necessary, cannot exceed 80. The files zmisc2.dat, zmisc15.dat, and zmisc16.dat are supplied with the dc5z tools. These are all the zmisc files that are needed.

8.10. **REAL32 background**. List here the 32 bit floating point number representing the intensity of the background in which the targets are embedded. This number is first defined in df.dat. Since the df codes deal with byte images, each byte representing an intensity, the real number here should be between 0.0 and 255.0. Note that there is less flexibility here than in the df codes, which can handle different backgrounds for each new target set. Since this floating point number is only used in the documentation file \temp\scra.tmp and not in any computation, future versions of the dc5 codes may omit this parameter completely.

8.11. **INT image.stride**. List here the 32 bit integer which is the stride through the target input files first defined in df.dat. Note that there is less flexibility here than in the df codes, which can handle a different stride for each new target set. Since this integer is only used in the documentation file \temp\scr.tmp and not in any computation, future versions of the dc5z codes may omit this parameter completely.

8.12. **INT label.of.first.false.target.used**. This 32 bit integer is first defined in the df.dat. If there is no false target set or if there is more than one false target set or if there is a single false target set read in the image label input mode, set this 32 bit integer to 0. If there is a single false target set read in the sequential input mode, then set this 32 bit integer to be the label of the very first image used in the false target file. Note that there is less flexibility here than in the df codes, which can handle different labels for each new target set. Since this integer is only used in the documentation file \temp\scr.tmp and not in any computation, future versions of the dc5z codes may omit this parameter completely.

8.13. **INT label.of.first.true.target.used.** This 32 bit integer is first defined and used in df.dat. If there is more than one true target set or if there is a single true target set but it is not read in the sequential input mode, set this 32 bit integer to 0. If there is a single true target set read in the sequential input mode, then set this 32 bit integer to be the label of the very first image used in the true target file. Note that there is less flexibility here than in the df codes, which can handle different labels for each new target set. Since this integer is only used in the documentation file \temp\scr.tmp and not in any computation, future versions of the dc5z codes may omit this parameter completely.

8.14. **[80]BYTE original.false.target.filename**. This linear byte array contains a list of the false target filenames, first defined in df.dat. They must be written on a single line no more than 80 characters long. For clarity the different names perhaps should be separated by commas or semicolons. Since this byte array is only used in the documentation file \temp\scr.tmp and not in any computation, future versions of the dc5z codes may omit this parameter completely.

8.15. **[80]BYTE original.true.target.filename**. This linear byte array contains a list of the true target filenames, first defined in df.dat. They must be written on a single line no more than 80 characters long. For clarity th: ferent names perhaps should be separated by commas or semicolons. Since this byte array is only used in the documentation file \temp\scr.tmp and not in any computation, future versions of the dc5z codes may omit this parameter completely.

8.16. **[80]BYTE target.input.method**. The choices to be written here are "standard amplitude input method", "standard intensity input method", "intensity phase-encoded input method", or "amplitude phase-encoded input method". Which choice to use may be gleaned from the appropriate booleans used in df.dat. Since this byte array is only used in the documentation file \temp\scr.tmp and not in any computation, future versions of the dc5z codes may omit this parameter completely.

8.17. **BOOL bool.shift.mode**. This boolean is first used in the df codes and discussed in df.doc. It is first defined in df.dat and should have the same value here.

8.18. **[80]BYTE command.line.2.0**. This command line should read "copy c:\temp\scr.tmp c:\temp\dscr5a2.tmp", for example. It is used to copy the first version of scr.tmp to more permanent storage. The number of characters in this command line cannot exceed 80. This command line has an effect only if the number of discretization states is 2.

8.19. **[80]BYTE command.line.2.1**. This command line should read "copy d:\temp\scr.tmp d:\temp\dscr5b2.tmp", for example. It is used to copy the final version of scr.tmp to more permanent storage. The number of characters in this command line cannot

exceed 80. This command line has an effect only if the number of discretization states is 2.

8.20. **[80]BYTE command.line.2.2**. This command line should read "copy d:\temp\pod.tmp d:\temp\di5b2.tmp", for example. It is used to copy the final version of pod.tmp to more permanent storage. The number of characters in this command line cannot exceed 80. This command line has an effect only if the number of discretization states is 2.

8.21. **[80]BYTE command.line.15.0**. This command line should read "copy c:\temp\scr.tmp c:\temp\dscr5a15.tmp", for example. It is used to copy the first version of scr.tmp to more permanent storage. The number of characters in this command line cannot exceed 80. This command line has an effect only if the number of discretization states is 15.

8.22. **[80]BYTE command.line.15.1**. This command line should read "copy d:\temp\scr.tmp d:\temp\dscr5b15.tmp", for example. It is used to copy the final version of scr.tmp to more permanent storage. The number of characters in this command line cannot exceed 80. This command line has an effect only if the number of discretization states is 15.

8.23. **[80]BYTE command.line.15.2**. This command line should read "copy d:\temp\pod.tmp d:\temp\di5b15.tmp", for example. It is used to copy the final version of pod.tmp to more permanent storage. The number of characters in this command line cannot exceed 80. This command line has an effect only if the number of discretization states is 15.

8.24. **[80]BYTE command.line.16.0**. This command line should read "copy c:\temp\scr.tmp c:\temp\dscr5a16.tmp", for example. It is used to copy the first version of scr.tmp to more permanent storage. The number of characters in this command line cannot exceed 80. This command line has an effect only if the number of discretization states is 16.

8.25. **[80]BYTE command.line.16.1**. This command line should read "copy d:\temp\scr.tmp d:\temp\dscr5b16.tmp", for example. It is used to copy the final version of scr.tmp to more permanent storage. The number of characters in this command line cannot exceed 80. This command line has an effect only if the number of discretization states is 16.

8.26. **[80]BYTE command.li 16.2**. This command line should read "copy d:\temp\pod.tmp d:\temp\di5b16 ", for example It is used to copy the final version of pod.tmp to more permanent storage. The number of characters in this command line cannot exceed 80. This command line has an effect only if the number of discretization states is 16.

9.0. **Editing the zmisc.dat File**. This file contains various constants controlling aspects of the optimal discretization process. The files zmisc2.dat, zmisc15.dat, and zmisc16.dat are samples of the zmisc.dat file which can be used for creating optimal discretized filters with 2, 15, and 16 equally spaced phase states, respectively. These three samples are all the zmisc files that are needed.

9.1. **REAL32 clock.multiplier**. This 32 bit floating point number should be set to 64.0.

9.2. **INT discretization.states**. This 32 bit integer is the number of discrete phase states in the output filter pod.tmp. For most purposes this number usually is chosen to be 2, 15, or 16.

9.3. **REAL32 false.target.threshold**. This 32 bit floating point number should be chosen to be 0.95.

9.4. **INT initial.zdenominator**. This 32 bit positive integer helps control the accuracy of the discretization. The number 512 is recommended.

9.5. **initial.length**. This 32 bit positive integer helps control the accuracy of the discretization. The number 512 is recommended. The parameter initial.length should be no more than initial.zdenominator.

9.6. **initial.znumerator**. This 32 bit nonnegative integer helps control the accuracy of the discretization. The number 0 is rec mend

9.7. **second.stage.multiplier**. This 32 bit nonnegative integer helps control the accuracy of the discretization. The number 64 is recommended.

9.8. **write.zdenominator.timing**. This boolean should be set to be TRUE.

9.9. **write.znumerator.timing**. This boolean should be set to be TRUE.

10.0. **Terminating the Program**. The program will terminate on its own, either successfully or in an erroneous state, in which case an error message will probably appear on the PC screen. The most common cause of an error message is the nonexistence of a file the program was seeking to use. The user can terminate the program by typing "control/break" and then typing "x".

11.0. **Cleaning up the \dc5z\ subdirectory**. The \dc5z\ subdirectory may be cleansed of the new files created by invoking the batch file cleanup.bat, i.e., by typing "cleanup". This is useful for housekeeping purposes. Be aware that dc5z.btl is deleted in this process.

APPENDIX H

THE HIGHLIGHTS OF THE GRAPHICS CODES

The author has designed several occam 2 graphics programs. The capabilities of the two most important of these, the gp1 and gp2, are sketched here.

The gp1 codes are a vast expansion of the first imaging program created by the author several years ago. They have a great many features, only a few of the more important ones being listed here. They can take a byte image, rotate it, scale it, translate it, put it into an arbitrary background, and add a variety of noises to the target and/or the background. They can then edge enhance the resulting image using the Sobel technique, a great variety of the diff techniques, or the unique local Wigner intensity phase-encoded method discussed in Section XVII. Finally, they can binarize the edge enhanced or original image using a plethora of independent local and/or global techniques. All three images (the modified original image, the edge enhanced image, and the binarized image) can be independently appended to an existing file or put into a new file. The gp1 codes have proved to be a great research and image processing tool.

The gp1 codes have been a great help in investigating which edge enhancement/binarization preprocessing technique for spatial filter training set preparation should be coupled with which edge enhancement/binarization preprocessing technique on the input imagery to a correlator. The author's tentative conclusion is somewhat surprising: the training set imagery should be edge enhanced with the local Wigner intensity phase-encoded method and then binarized using the global histogram method; the method to be employed on the input imagery to a correlator is the Sobel edge enhancement technique together with the global histogram binarization method.

The gp2 simulation codes enable one to test the performance of a wide variety of spatial filters designed for a very wide variety of inputs. All of the image processing techniques incorporated in the current gp1 codes are included as options for processing the input imagery in the gp2 codes. The gp2 codes permit the user to view the correlation plane output of a byte image, processed as specified by the user, versus a spatial filter. Various screens show the input image and/or its edge enhanced version and/or its binarized version, the south-

to-north view of the correlation plane, the west-to-east view of the correlation, a variety of top down views of the correlation plane, a histogram of the nonzero pixels of the target, the power spectrum of the input image in a variety of modes, the impulse response of the filter, and the filter itself. These simulation codes have been of immense use to the author in his work and research.

APPENDIX I

THE AUTHOR'S COMPUTING ENVIRONMENT

All of the computations described in this report were done very quickly and efficiently on two nearly identical low cost systems, one located in the author's office and the other located in the author's apartment. Each system uses an 66-megahertz Intel 486/DX2 PC with a 340 megabyte hard disk as a host. The computations themselves were done using a variety of INMOS and Transtech transputer modules which essentially reside on some boards in these PC's. A server program runs on the PC during the computations. Though one could have written the necessary software using either C, Fortran, or even ADA toolsets, the author chose to write his design codes in the parallel programming language occam 2. This language occam 2 is very simple and easy to use and is the best way to extract the most performance from transputers since it was designed simultaneously with the hardware. The occam 2 D7305A toolset was the specific one used. All of the figures in this report are photographs of the screen of a SONY GDM-1953 color graphics monitor in the author's office. This monitor is driven by a Transtech TTG3-8 graphics transputer module. The author chose to write from scratch in occam 2 the software used by this graphics transputer module to drive the monitor. These transputer based systems are relatively old and inefficient by modern standards. In the near future the author hopes to transfer his spatial filter design codes and simulation codes to ANSI standard *C* for use in modestly priced modern workstations, where the spatial filter design times should be reduced by a factor of ten.

APPENDIX J

COMPUTATIONAL PRACTICALITY

There seems to be some question in the literature as to the computational practicality of the phase-only filter design techniques invented by the author. Examples in this regard are: "Computing the solution to the minimax problem (as suggested by Kallman) requires enormous amounts of computer time, and an exhaustive search must be carried out to optimize the proposed criterion" (A. Mahalanobis et al., Reference 31); "...this method is not generally applicable because of its enormous computation load, which requires a supercomputer" (Kumar et al., Reference 32); and "...this problem does not lend itself easily to mathematical analysis and seems to require an exhaustive search which results in huge computational requirements that may not be met when the number of training images is large" (Reference 33, Arsenault et al.). Statements of a similar ilk can be found in Reference 34. It is rather astonishing that such statements appear in what purport to be refereed journals. Such statements are nothing but the personal opinions of the authors unsubstantiated by any evidence. This must be the case since these statements are simply false. Well defined algorithms do exist to solve the optimization questions involved in the author's approach to spatial filter design. These are detailed in Section IX and Section X of this report. This algorithms can be turned into efficient computer codes as demonstrated by the author's design codes. For a more concrete example of the efficacy of these design codes can be found in Table 1. Table 1 gives a sequence of strobes of time and intermediate values of SCR for the first 500 seconds of the design of an optimized continuous phase-only filter with a training set of 91 true target M48 tank images. The computation times and improvements in SCR listed in Table 1 are typical. Moreover, in a recent series of design runs the author used his systems to design 270 spatial filters whose entries are either 0, +1, or -1. The training sets for these filters consisted of 45 true binarized targets. The total design times for these filters invariably ranged between ten and fifteen minutes for each filter. Thus the ex cathedra assertions that one needs supercomputers to implement the author's approach to filter design are simply false. As mentioned in Appendix I, a decrease in computing time by a factor of ten is expected when the author's spatial filter design codes are transferred to modern workstations. The

ludicrous statements discussed here were first parried in Reference 23.

TABLE 1. A Sequence of Strobes into the Optimization Calculation for a Continous Phase-Only Filter with a Training Set of 91 True Targets

Seconds into the Optimization	Intermediate SCR's
0	2.49
44	15.86
125	19.88
188	24.07
253	26.96
325	30.06
399	33.47
497	36.26

REFERENCES

1. M.I. Skolnik, Radar Handbook, McGraw-Hill, 1970.

2. W. Feller, "On the logistic law of growth and its empirical verifications in biology," Acta Biotheoretica, volume 5, 1940, pp. 51 — 66.

3. J.W. Goodman, Introduction to Fourier Optics, McGraw-Hill, 1968.

4. J.L. Horner and P.D. Gianino, "Phase-only matched filtering," Applied Optics, volume 23, number 6, 15 March 1984, pp. 812 — 816.

5. M. Born and E. Wolf, Principles of Optics, sixth (corrected) edition, Pergamon Press, 1987.

6. D. Casasent and H.J. Caulfield, "Basic Concepts," Optical

Data Processing Applications, Springer-Verlag, 1978.

7. J.D. Gaskill, Linear Systems, Fourier Transforms and

Optics, Wiley, 1978.

8. S.H. Lee, "Basic Principles of Optical Information Processing," Optical Information Processing Fundamentals, Springer-Verlag, 1981.

9. G.O. Reynolds, J.B. DeVelis, G.B. Parrent, Jr., and B.J. Thompson, The New Physical Optics Notebook: Tutorials in

Fourier Optics, copublished by SPIE — The International Society for Optical Engineering and the American Institute of Physics, 1989.

10. H.J. Wea Applications of Discrete and Continuous Fourier

Analysis, Wiley-Interscience, 1983.

11. R.R. Kallman, Construction of low noise optical correlation filters, Applied Optics, volume 25, number 7, 1 April 1986, pp. 1032 — 1033.

12. B.V.K. Vijaya Kumar, Tutorial survey of composite filter designs for optical correlators, Applied Optics, volume 31, number 23, 10 August 1992, pp. 4773 — 4801.

13. R.R. Kallman, The construction of a general spatial filter as a constrained optimization question, Proceedings of SPIE - The International Society for Optical Engineering,

volume 1959, Optical Pattern Recognition, pp. 104 — 118. Proceedings of conference held 12-16 April 1993, Orlando, Florida. David P. Casasent, Chairman/Editor.

14. R.R. Kallman, Direct construction of phase-only correlation filters, Proceedings of SPIE — The International Society for Optical Engineering, volume 827, Real Time Signal Processing X, pp. 184 — 190. Proceedings of conference held 20-21 August 1987 in San Diego, California. J. P. Letellier, Chairman/Editor.

15. R.R. Kallman, Direct construction of phase-only filters, Applied Optics, volume 26, number 24, 15 December 1987, pp. 5200 — 5201.

16. D. A. Jared, Distortion range of filter synthetic discriminant function binary phase-only filters, Applied Optics, volume 28, 1989, pp. 4835 — 4839.

17. R.R. Kallman, The Design of Phase-Only Filters for Optical

Correlators, AFATL-TR-90-63, Air Force Armament Laboratory, Eglin Air Force Base, Florida 32542, July 1990, pp. i-xii +1-104.

18. R.R. Kallman, The Design of Phase-Only Filters for Millimeter Wavelength Radar Imagery, WL-TR-93-7082, Wright Laboratory, Eglin Air Force Base, Florida, August 1993, pp. i — xxii +1-180.

19. G.B. Dantzig, Linear Programming and Extensions, Princeton University Press, 1963.

20. R.R. Kallman, An algorithm to choose which pixels to zero out in a zero and phase filter, Proceedings of SPIE — The International Society for Optical Engineering, volume 2026, Photonics for Processors, Neural Networks, and Memories — Optical Implementation of Information Processing, pp. 167 — 176. Proceedings of conference held 11-16 July 1993, San Diego, California. Joseph L. Horner and Bahram Javidi, Chairmen/Editors.

21. D.L. Flannery, J.S. Loomis, and M.E. Milkovich, Transform ratio ternary phase-amplitude filter formulation for improved correlation discrimination, Applied Optics, volume 27, number 19, 1 October 1988, pp. 4079 — 4083.

22. R.R. Kallman, Optimal low noise phase-only and binary phase-only correlation filters for threshold detectors, Applied Optics, volume 25, number 23, 1 December 1986, pp. 4216 — 4217.

23. R.R. Kallman and D.H. Goldstein, Invariant phase-only filters for phase-encoded in-

puts, Proceedings of SPIE — The International Society for Optical Engineering, volume 1564, Optical Information Processing Systems and Architectures III, pp. 330 — 347. Proceedings of conference held 20-25 July 1991, San Diego, California. Bahram Javidi, Chairman/Editor.

24. R.R. Kallman, Invariant phase-only filters for TABILS24 millimeter wavelength radar data, Proceedings of SPIE — The International Society for Optical Engineering, volume 1960, Automatic Object Recognition III, pp. 74 — 90. Proceedings of conference held 12-16 April 1993, Orlando, Florida. Firooz A. Sadjadi, Chairman/Editor.

25. A.V. Oppenheim and J im, The importance of phase in signals, Proceedings of the IEEE, volume 69, number 5, May 1981, pp. 529 — 541.

26. R.R. Kallman and D.H. Goldstein, Phase-encoding input images for optical pattern recognition, Optical Engineering, volume 332, number 6, June 1994, pp. 1806 — 1812.

27. R.R. Kallman and D.H. Goldstein, United States Patent Number 5,214,534, May 25, 1993, Coding Intensity Images as Phase-Only Images for Use in an Optical Correlator. A synopsis of this patent appeared in the Official Gazette of the United States Patent and Trademark Office, volume 1150, number 4, 25 May 1993, p. 2664.

28. R.R. Kallman, Two variants of the optical correlation process, Proceedings of SPIE — The Internati al Society for Optical Engineering, volume 938, Digital and Optical Shape Representation and Pattern Recognition, pp. 40 — 47. Proceedings of conference held 4-8 April 1988, Orlando, Florida. Richard D. Juday, Chairman/Editor.

29. R.R. Kallman, Invariance properties of spatial filters designed for zero mean inputs, Proceedings of SPIE — The International Society for Optical Engineering, volume 2297, Optical Implementation of Information Processing, pp. 40 — 51. Proceedings of Conference held 24-29 July 1994, San Diego, California.

30. F. Hlawatsch and G.F. Boudreau-Bartels, Linear and quadratic time-frequency signal representations, IEEE Signal Processing Magazine, volume 9, number 2, April 1992, pp. 21 — 67.

31. A. Mahalanobis, B.V.K. Vijaya Kumar, and D. Casasent, et al., Minimum average correlation energy filters, Applied Optics, volume 26, 1987, pp. 3633 — 3640.

32. B.V.K. Vijaya Kumar, Z. Bahri, and A. Mahalanobis, Constraint phase optimization in minimum variance syuthetic discriminant functions, Applied Optics, volume 27, number

2, 15 January 1988, pp. 409 — 413.

33. A. Metioni, H.H. Arsenault, and L. LeClerc, Methods for reducing sidelobes associated with composite filters, Optics Communications, volume 71, number 6, 5 June 1989, pp. 332 — 336.

34. J. Ding, M. Itoh, and T. Yatagai, Iterative design of distortion-invariant phase-only filters with high Horner efficiency, Optical Engineering, volume 33, number 12, page 4037 — 4044.

BIBLIOGRAPHY

1. D.L. Flannery, J.S. Loomis, and M.E. Milkovich, Transform ratio ternary phase-amplitude filter formulation for improved correlation discrimination, Applied Optics, volume 27, number 19, 1 October 1988, pp. 4079 — 4083.

2. R.W. Hamming, Digital Filters, second edition, Prentice-Hall, Englewood Cliffs, New Jersey, 1983.

3. J.L. Horner and P.D. Gianino, Applying the phase-only filter concept to the synthetic discriminant correlation filter, Applied Optics, Volume 24, 1985, p. 851.

4. R.R. Kallman, The optimal construction of synthetic discriminant functions for optical matched filters, Report # 73, pp. 1 — 28, United States Air Force Summer Faculty Research Program, 1984, Technical Reports, Volume II, The SCEEE Press, Orlando, Florida, 1985.

5. R.R. Kallman, The optimal construction of synthetic discriminant functions (SDF's), U.S. Air Force Armament Laboratory Technical Report AFATL-TR-86-19, April 1986, pp. i-vi+1-24.

6. R.R. Kallman, The construction of low noise optical correlation filters and their application to target identification problems, U.S. Air Force Armament Laboratory Technical Report AFATL-TR-86-20, July 1986, pp. i-vi+1- 41.

7. N. Levanon, Radar Principles, Wiley-Interscience, 1988.

8. S. Lawrence Marple, Jr., Digital Spectral Analysis with Applications, Prentice-Hall, Englewood Cliffs, New Jersey, 1987.

9. Alan V. Oppenheim and Ronald W. Schafer, Digital Signal Processing, Prentice-Hall, Englewood Cliffs, New Jersey, 1975.

10. D. Psaltis, E.G. Paek, and S.S. Venkatesh, Optical image correlation with a binary spatial light modulator, Optical Engineering, volume 23, number 6, November/December 1984, pp. 698 — 704.

11. W.E. Ross, D. Psaltis, and R.H. Anderson, Two-dimensional magneto-optic spatial light modulator for signal processing," Optical Engineering, volume 22, number 4, July/August 1983, pp.485 — 490.

12. M.I. Skolnik, Introduction to Radar Systems, McGraw-Hill, 1980.

13. G.W. Stimson, Introduction to Airborne Radar, Hughes Aircraft Company, 1983.

DISTRIBUTION

DEFENSE TECHNICAL INFORMATION CENTER ATTN: DTIC-DDAC 8725 JOHN J. KINGMAN ROAD, SUITE 0944 FT. BELVOIR VA 22060-6218	2
WL/CA-N DR. S. LAMBERT 101 W EGLIN BLVD, SUITE 105 EGLIN AFB FL 32542-5434	1
SCIENTIFIC AND TECHNICAL INFORMATION FACILITY 203 W EGLIN BLVD, SUITE 300 EGLIN AFB FL 32542-6843	1
WL/MNGA DR. D. GOLDSTEIN 101 W EGLIN BLVD, SUITE 280 EGLIN AFB FL 32542-6810	5
WL/MNG 101 W EGLIN BLVD, SUITE 268 EGLIN AFB FL 32542	2
NAVAL AIR WEAPONS CENTER ATTN: DR. DAVE BURDICK CODE C2901A CHINA LAKE CA 93555-6001	1
COMDR, U.S. ARMY MISSILE COMMAND]ATTN: AMSMI-RD-GC-N (MR. JIM McCLEAN) REDSTONE ARSENAL AL 35898-5241	1
IIT RESEARCH INSTITUTE/GACIAC 10 WEST 35 STREET CHICAGO IL 60616-3799	1
DR ROBERT R. KALLMAN UNIVERSITY OF NORTH TEXAS MATHEMATICS DEPARTMENT PO BOX 5116 DENTON TX 76203-5116	1

(The reverse of this page is blank)